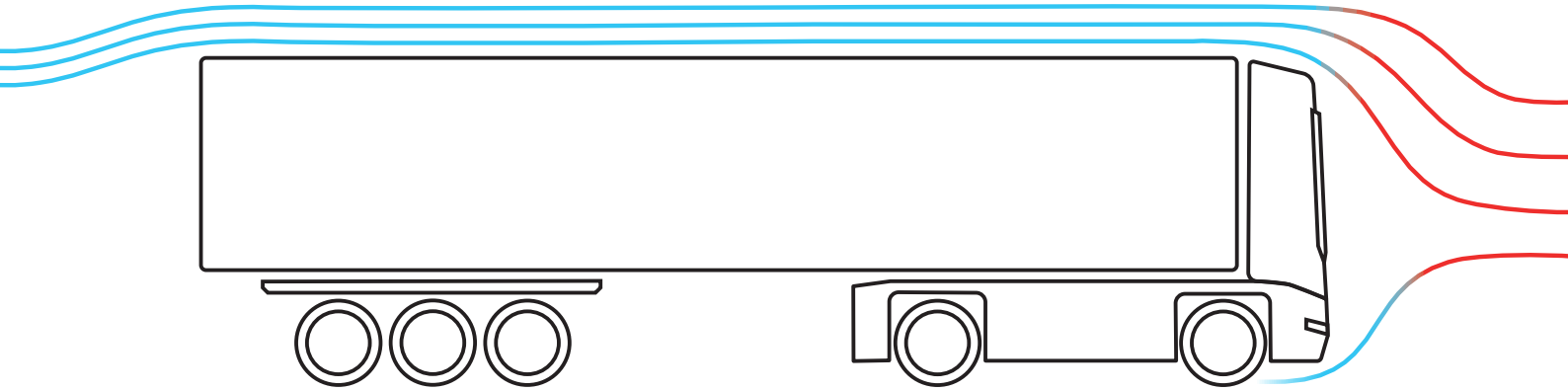
Dietrich Steinmetz

New Heuristics for Finding Optimal Routes for Vehicle Platoons in Road Networks

Doctoral Thesis



Germany, 2020

Dietrich Steinmetz

New Heuristics for Finding Optimal Routes for Vehicle Platoons in Road Networks

Doctoral Thesis

to be awarded the degree of

Doctor rerum naturalium (Dr. rer. nat.)

at the

Faculty of Mathematics, Computer Science, and Mechanical Engineering
Clausthal University of Technology

February 21, 2020

Supervisor and First Examiner: Prof. Dr. Sven Hartmann
Second Examiner: Prof. Dr. Stephan Westphal

Date of the Oral Examination: April 20, 2020
Chairman of the Board of Examiners: Prof. Dr. Jörg Müller

Declaration of Authorship

I, Dietrich Steinmetz, declare that this thesis titled, "New Heuristics for Finding Optimal Routes for Vehicle Platoons in Road Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: February 21, 2020

Foreword

My parents, *Irina* and *Ivan Steinmetz*, taught me the importance of education and science at a young age. Over the years, I developed a strong interest in exploring and researching the world, and my parents have lovingly supported me. For that, I would like to thank my parents for everything.

In my academic career, I have met many interesting and inspiring people. One of these people is my doctoral advisor, *Prof. Dr. Sven Hartmann*. The years of working together have been much fun and very productive. In stressful situations, I could always count on your support. Thank you very much for making this dissertation possible. I would also like to thank my second advisor, *Prof. Dr. Stephan Westphal*, for his great scientific input, the numerous discussions, and of course, for the delicious coffee.

Gerrit Burmester has been accompanying me for many years, both as a friend and colleague. Together in the same office, we have developed and realized many ideas. Specifically, his point of view as an engineer was very valuable, thank you very much for the great support. Another friend, *Felix Merz*, takes a close look at things. During the countless evenings, numerous ideas were worked out, from complex mathematical concepts to the structure of a cake; all of this with great attention to detail — thank you very much. I would also like to thank *Daniel Dyballa* for entertaining and productive evenings; among other things, this resulted in a terrific publication. In general I thank my friends for being there for me and for their encouragement. I was often swamped when working on this dissertation, and sometimes I went underground for a longer time.

For the detailed review and proofreading of this work, I would like to thank *Evgeny Steinmetz*, *Dr. Andreas Reinhardt*, *Marc Timmermann*, and *Erin Renee Flanagan*. I thank *Jessica Lütge* for her support in designing the cover of this work. I would also like to thank all the students I have worked with or supervised. I am grateful for *Pascal Kleindienst* for creating a system for conducting the experiments as part of his master thesis. Special thanks also go to my colleagues at the Institute of Computer Science for a great working atmosphere and successful cooperation.

"If everyone is honest, the exchange of ideas - is a completely natural phenomenon. But only if everyone is honest."

Grigori Perelman

Abstract

Faculty of Mathematics, Computer Science, and Mechanical Engineering
Clausthal University of Technology

New Heuristics for Finding Optimal Routes for Vehicle Platoons in Road Networks

by Dietrich Steinmetz

Platooning is the coupling of two or more trucks with small inter-vehicle distance. This technology has the potential to improve road safety, to increase road capacity, and to reduce CO_2 emissions. This thesis focuses on minimizing the overall fuel consumption by the centralized formation of vehicles into platoons. Several novel solution strategies are developed for efficiently and effectively tackling large-scale problem instances. One of these strategies stipulates the grouping of vehicles with high platooning incentives and subsequent computation of platooning routes for each vehicle group independently of the other vehicles. This work proposes three lightweight incentive methods based on inexpensive geometric operations. Several grouping approaches, which are using these incentives, are developed to coordinate the vehicles and compared against each other for their efficiency and effectiveness. Our experimental evaluation demonstrates that our proposed grouping-based routing algorithms are more scalable than exact solvers and state-of-the-art routing methods for vehicle platooning.

Furthermore, this thesis introduces a new efficient Column Generation formulation for the Vehicle Platooning Problem. An equivalent path-based Master Problem (MP) is given for the Vehicle Platooning Problem (VPP), where the number of variables is much larger than the number of constraints. The Column Generation method restricts and solves the MP iteratively. The corresponding subproblems generate new promising platooning paths and add them to the Restricted Master Problem (RMP). The RMP can be solved very efficiently as only a sufficiently small subset of variables is considered at a time.

The composition of our grouping algorithms and our Column Generation (CG)-based routing algorithm enables us to solve extensive problem instances with up to **2000 vehicles** on road networks with up to **300,000 edges**.

Contents

Dedicated to science.

Chapter 1

Introduction

1.1 Motivation

Platooning is the linking of two or more vehicles with small inter-vehicle distances with automatic control, as shown in Fig. 1.1. This innovative technology optimizes the safety of road transportation [47, 16], increases the capacity of road networks [16], and reduces the CO_2 emissions [11, 7]. In the European Union, 23% of the total CO_2 emissions in 2017 were caused by road transportation [67]. According to the International Transport Forum (ITF) [28], the total freight transport will grow by a factor of 1.6 between 2017 and 2050. Vehicle platooning is an important technology to reduce the CO_2 emissions in road transportation by decreasing the vehicle aerodynamic drag [94]. The recent research investigated the aerodynamic behavior of vehicles in platoons and corresponding fuel consumption. The results showed significant improvements in the fuel savings [34, 85, 53]. However, different factors such as inter-vehicle distance, vehicle speed, and vehicle load influence fuel savings [52].

FIGURE 1.1: Trucks driving with small inter-vehicle distances to reduce the total fuel consumption [91].

This work deals with the fuel-efficient platooning routing problem. Many works focus on fuel-optimal routing for the existing platoons or vehicle routes. The major challenge here is the fuel-optimal speed adjustment and vehicle scheduling. Furthermore, much research has been done in the field of decentralized platooning, where the vehicles spontaneously form the platoons. In this case, the vehicles drive within a short distance to each other and dynamically coordinate themselves. The centralized platooning provides higher savings potential than the local approaches. Fig. 1.2 illustrates an example of centralized platooning. *Vehicle A* travels from *Tallahasse, Florida* to *Washington D.C.*, and *Vehicle B* travels from *Miami, Florida* to *Winchester, Virginia*. Centralized platooning includes an optimal merging point at *Jacksonville,*

Florida. The common platooning path of *Vehicle A* and *Vehicle B* is shown as a green dotted line. The vehicles split up close to their destinations. Without the centralized coordination, *Vehicle A* would take its shortest path (red dotted line) and waste possible platooning potential. In this case, the spontaneous platooning opportunity is minimal, and the detour of a few kilometers reduces the fuel consumption of both vehicles enormously. [57] formalized the vehicle platooning problem and showed that the problem is NP-hard even when the vehicle's deadlines are not considered. The complexity of the road network plays a significant role in the computation of the platooning routings. The vast road networks with complex topologies increase the platooning problem's difficulty, due to the increasing number of potential platooning paths. Increasing the number of vehicles increases the computing complexity and platooning opportunities.

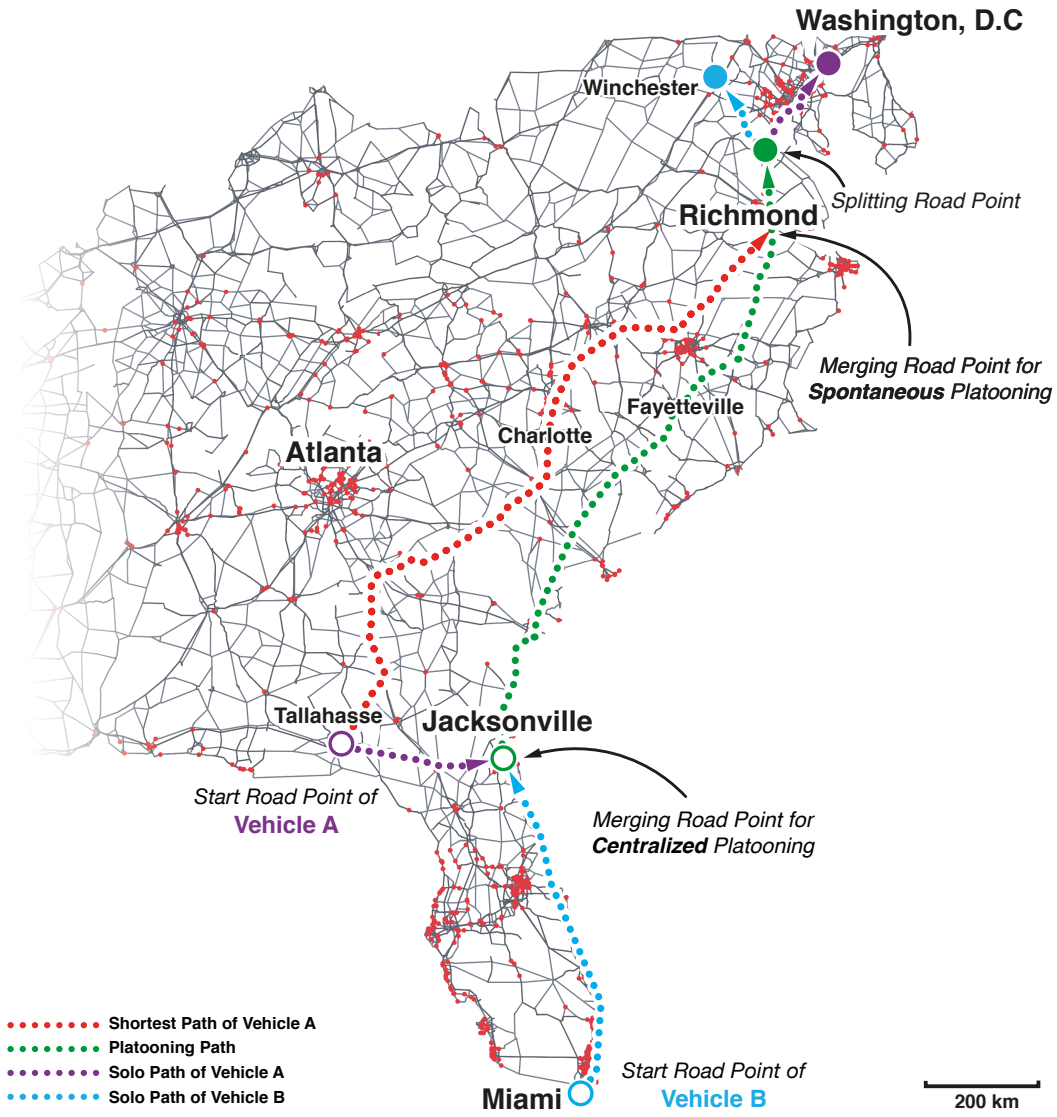


FIGURE 1.2: Example of the centralized and spontaneous platooning.

This research addresses the fuel-optimal platooning routing problem. The primary goal is to identify efficient methods to tackle large-scale problem instances with complex road networks and a large number of vehicles.

1.2 Structure and Content

This thesis is structured as follows. Chapter 2, *Preliminaries*, describes the necessary foundations and formally introduces the Vehicle Platooning Problem with corresponding basic definitions. Chapter 3, *State of the Art*, discusses relevant literature and categorizes the state-of-the-art and related works. The research questions, contributions, and limitations are proposed in Chapter 4, *Research Goal and Contributions*. Chapter 5, *Heuristics for Finding Near-Optimal Groups for Vehicle Platooning*, introduces our grouping approaches, these decompose the original problem into smaller subproblems and solve them efficiently. Chapter 6, *Computation of near-optimal Platoon Routing based on the Candidate Groups*, proposes the novel formulation of the CG algorithms for the VPP. This chapter also contains the implementations of state-of-the-art algorithms *Best Pair* and *Hub Heuristic*. Chapter 7, *Graph Data Model for Graph Database Support of Platooning*, deals with the problem-relevant data. For this purpose, the labeled graph property model is formulated, which organizes VPP data elements. Furthermore, this chapter describes the architecture of the VPP framework, modularizing the proposed algorithms in this work. The conducted computational experiments are described in Chapter 8, *Experimental Evaluation*. Chapter 9, *Conclusions and Future Research*, concludes the entire work and describes future work.

Chapter 2

Preliminaries

This chapter provides formal definitions of basics concepts. Section 2.1 formally introduces the Vehicle Platooning Problem. Section 2.1 contains an overview of important notations used in this work.

2.1 Vehicle Platooning Problem (VPP)

A road network is a directed graph $G = (V, E)$ with a finite set of nodes V and a set of edges $E \subseteq V \times V$. The nodes of G represent the *road points*, and the edges of G represent the *road segments*. The coordinate-function $\gamma : v \mapsto \mathbb{R}^2$ provides *latitude* γ_1 and *longitude* γ_2 coordinates for all $v \in V$. An edge is defined as follows $e = (v, u), v \in V, u \in V : v \neq u$. The weight-function $w : e \mapsto \mathbb{R}_+$ provides a deterministic distance for all $e \in E$. A travel-cost-function $c : e \mapsto \mathbb{R}_+$ assigns probabilistic travel costs to each edge, which are determined by $0 \leqslant c(e) = w(e)$. In real road networks, the values of these functions differ.

The Eq. 2.1 defines a vehicle as a tuple with the start a and the destination b nodes in graph G. Additionally, each vehicle includes the earliest departure t^a and the latest arrival t^b deadlines. The set of vehicles is defined in Eq. 2.2 and is used uniformly in this work.

$$h = (a, b, t^a, t^b) \in V \times V \times \mathbb{Z}_+ \times \mathbb{Z}_+ : a \neq b \ \wedge t^a < t^b \tag{2.1}$$

$$H = \{h_1, \ldots, h_{|H|}\} \tag{2.2}$$

The vehicle path or route, Eq. 2.3 is an ordered list of tuples, where each tuple (e, t) consists of an edge e and the earliest arrival time t at that edge. This way, the shortest, alternative, and platooning routes can be described. This work assumes constant travel speed $\vartheta = 1$ of vehicles. The minimum edge traversal time is defined by Eq. 2.4; the difference between the right-hand side and the left-hand side implies a waiting time. The nodes in path occur exactly once so that cycles are not allowed.

$$P = \{((v_1, u_1), t_1) \ldots ((v_q, u_q), t_q)\}, (v_i, u_i) \in E, t_i \in \mathbb{Z}_+ \ \forall i = 1, \ldots, q \tag{2.3}$$

$$t_i + \frac{w(e_i)}{\vartheta} \leqslant t_{i+1} \ \forall i = 1, \ldots, q-1 \tag{2.4}$$

A travel-cost-function $c : P \mapsto \mathbb{R}_+$ is overloaded and maps probabilistic travel costs to any feasible path P.

Definition 2.1.1 (Vehicle Platoon)**.** A vehicle platoon (or platoon for short) is a subset of the vehicle set H, with at least two vehicles. The routes of vehicles, which belongs

to the same platoon, have at least one common edge and **are not disjoint**. The costs of non-disjoint edges for given vehicles are reduced by the saving factor η, except for the leader vehicle's edges.

Definition 2.1.2 (Platooning Route)**.** The platooning route P_h^{pl} is a feasible path of vehicle h, which complies with the constraint 2.5.

$$c(P_h) \leq c(P_h^{pl}) \leq c(P_h) \cdot \frac{1}{1-\eta} \tag{2.5}$$

The lower bound of the platooning route costs $c(P_h^{pl})$ are the costs of the shortest path $c(P_h)$, and the upper bound costs are limited by a $c(P_h) \cdot \frac{1}{1-\eta}$.

Definition 2.1.3 (Platoon Routing)**.** The platoon routing is a set O of platooning routes with $|H| \geqslant 2$, as shown in Eq. 2.6. The costs of platoon routing are defined in Eq. 2.7.

$$O = \{P_1^{pl} \dots P_{|H|}^{pl}\} \tag{2.6}$$

$$c(O) = \sum_{(e,t) \in \bigcup O} \eta \cdot c(e) + \sum_{P \in O} \sum_{(e,t) \in P} (1-\eta) \cdot c(e) \tag{2.7}$$

This work assumes 10% fuel savings by reducing the air-drag. This saving factor in this work is defined as follows $\eta = 0.1$. The first term purchases all used edges in the routing O once. The second term adds up the reduced fuel consumption costs of all vehicle paths in O. The platoon routing is optimal if no further platoon routing with less cost exists for the given vehicle set H.

2.1.1 Integer Linear Program Formulation of VPP

This section formulates an Integer Linear Program for the Vehicle Platooning Problem in Eq. 2.8 - Eq. 2.12 based on [57]. The linear objective function minimizes the vehicles' travel costs, with consideration of the saving factor η. The binary decision variable y_e is 1 if any vehicle traverses edge e. The variable x_{eh} is 1 if vehicle h traverses edge e. The constraint of 2.9 enforces the flow conservation, the sum of incoming and outgoing edges is 1 if v is a start node, -1 if v is a destination node and otherwise 0. The constraint (2.10) ensures that the variable y_e is set to 1 when a vehicle uses the edge e.

$$\text{min} \quad \sum_{e \in E} \eta \cdot c(e) \cdot y_e + \sum_{e \in E, h \in H} (1-\eta) \cdot c(e) \cdot x_{eh} \tag{2.8}$$

$$\text{s. t.} \quad \sum_{e \in \delta^+(v)} x_{eh} - \sum_{e \in \delta^-(v)} x_{eh} = \begin{cases} 1 & \text{if } v = a_n \; \forall v \in V, 1 \leqslant n \leqslant |H| \\ -1 & \text{if } v = b_n \; \forall v \in V, 1 \leqslant n \leqslant |H| \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

$$x_{eh} \leq y_e \qquad \forall e \in E, h \in H \tag{2.10}$$

$$x_{eh} \in \{0,1\} \qquad \forall e \in E, h \in H \tag{2.11}$$

$$y_e \in \{0,1\} \qquad \forall e \in E \tag{2.12}$$

The Fig. 2.1 shows a schematic example of the Vehicle Platooning Problem with two vehicles $h = (1, 2, 0, \infty)$ and $p = (5, 6, 0, \infty)$ where $c(P_h) = 30$ and $c(P_p) = 30$.

The respective shortest routes have no common edges so that the vehicles achieve no fuel savings in terms of platooning. The right figure shows the optimal platoon routing with $P_h^{pl} = \{((1,3),0),((3,4),1),((4,2),30)\}$ and $P_p^{pl} = \{((5,3),0),((3,4),1),((4,6),30)\}$. The total cost of the platooning routing is 59.1 with 0.9 relative fuel-savings.

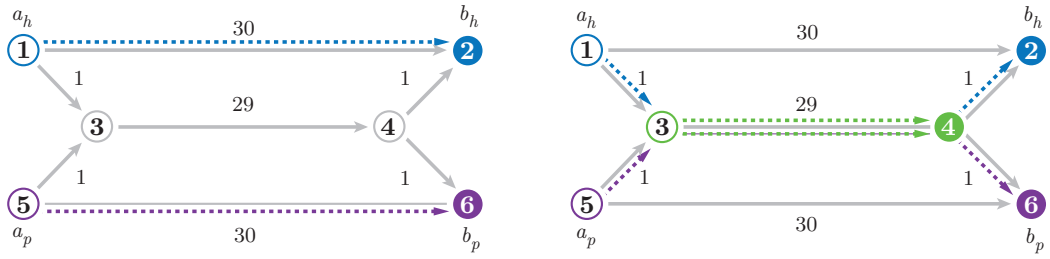


FIGURE 2.1: The schematic example of the VPP with two vehicles.

Table 2.1 summarizes all important notations. The listed notations are used consistently in the entire work.

TABLE 2.1: Important Notations

Notation	Description
$G = (V, E)$	Road network definition as a graph
$E \subseteq V \times V$	Road segments definition as a edge
V	Road points definition as a set of nodes
$e = (v, u), v \in V, u \in V : v \neq u$	Road segment definition as an edge
$c : e \mapsto \mathbb{R}_+$	Travel cost function of a road segment
$c : P \mapsto \mathbb{R}_+$	Travel cost function of a road segment route
$h = (a, b, t^a, t^b) \in V \times V \times \mathbb{Z}_+ \times \mathbb{Z}_+ : a \neq b \ \wedge t^a < t^b$	Vehicle definition with start and destination nodes and time restrictions
$H = \{h_1, \ldots, h_{\lvert H \rvert}\}$	Set of vehicles
$\gamma : v \mapsto \mathbb{R}^2$	Geographic coordinates of a road point
$\underline{h} = (\gamma(b) - \gamma(a))$	Vehicle direction vector
φ	Angle between two vehicle directions
$\eta = 0.1, \eta' = 0.5$	Fuel saving factor, linear fuel saving factor
$\lambda : H \times H \mapsto [0, 1]$	Platooning incentive function
$\Gamma(P_h)$	Geometric container surrounding the shortest path
$\tilde{G}$	Vehicles incentives graph
$\tilde{C}$	Vehicle group defined as a subclique of graph $\tilde{G}$
$\tilde{S}$	Set of vehicle groups
$\tilde{w} : \tilde{C} \to \mathbb{R}_+$	Weight of the vehicle group

Chapter 3

State of the Art

This chapter summarizes the state-of-the-art in the field of platoon routing. Firstly, there is an introduction to the developed classification of the VPP with an accordingly assignment of the existing works. Furthermore, Section 3.2 reviews some related routing problems and approaches.

3.1 Literature Review: Vehicle Platooning Problem

This section reviews the existing literature in the field of platooning routing and divides it into three classes.

3.1.1 Existing Platooning Routing or Shortest Path based Platooning Routing

The approach in [43] extracts different features from the shortest paths and time constraints of the vehicles. The features are used to discard the vehicle pairs from the detailed pairwise vehicle comparison. As a result, the approach provides platooning opportunities for vehicle assignments. [92] determines the vehicle's shortest paths and adjusts the speed on the individual edges with consideration of the time restrictions to minimize the fuel consumption. The vehicles can platoon only on intersecting edges of the considered shortest paths. [42, 44] compute the fuel-efficient plans with the corresponding shortest paths and vehicle speeds, taking the time restrictions into account. [60] investigates whether it is beneficial to drive faster to catch up with a vehicle and form the platoon accordingly. [93] provides an algorithm to compute fuel-optimal speed profiles for vehicles. Initially, the algorithm determines pairwise fuel-optimal plans, based on the shortest path of the vehicles. Each pair consists of a leader and a follower. The leader keeps a constant speed while the follower needs to adjust speed to merge with the leader. In the platoon phase, the follower has the same speed as the leader. After splitting up, the follower adjusts speed to meet its deadline. The fuel consumption and the potential fuel savings deviate from speed profiles. The result is defined as a directed, weighted graph, also referred to as a coordination graph. Nodes represent vehicles, and edges between nodes (vehicles) represent the ability to platoon. The weight of the edges (i, j) represents the fuel-saving of vehicle (i), if it follows vehicle (j). The partitioning around medoid's clustering algorithm is used to select leaders with the greatest fuel potential from the coordination graph. Generally, the algorithm is divided into a build-phase and a swap-phase. In the build-phase, the algorithm selects a random leader or a leader with the highest savings. In the swap-phase, there are also two different methods available to measure the gain of adding/removing a leader to/from the leader pool N_l. The algorithm halts, if no further improvements can be found.

A randomly generated road network with 100 nodes was used for the experiments. Start and destination nodes are selected randomly from a subset of 10 nodes. The number of vehicles K is defined in the range of 0 to 7000. For the first 1000 vehicles that formed platoons, the savings grow until 6%, afterwards the savings converge at 9%. The algorithm is based on existing routes for each vehicle. Platoons can be formed only on common parts of the road because the algorithm does not adjust any routes. The experiments were done with randomly generated data. The road network instance contains only 100 nodes, and the scalability of the savings in larger networks has not been tested. Also, the performance and complexity of the algorithm have not been mentioned.

3.1.2 Decentralized Platooning

[50, 56] introduce a local controller approach that only considers vehicles in the vicinity for the platoon formation. The local controller adjusts the speed and calculates the platoon routing of the respective vehicles. The experiments are conducted on a road network with 647 nodes and 12 destinations.

3.1.3 Centralized Platooning

The authors in [57] prove the NP-completeness of the platooning problems in general and planar graphs. The proof in general graphs includes the reduction of the platooning problem to the *set cover problem*. Since the authors do not consider the vehicle deadlines, they assume that time restrictions make platooning problems more complicated. The NP-completeness of the platooning problem is shown on planar graphs by reducing the *Rectilinear Steiner Arborescence Problem* (RSAP), which is also NP-complete. The planar graphs have the highest similarity to road networks.

[57] formulates the platooning problem as two various integer linear programs. In the first ILP, named *Unlimited Platooning Problem - Shared Starting Node*, all vehicles start on the same node, without any time constraints. Several vehicles traverse a certain edge always at the same time and at most in one platoon. In the next Integer Linear Program (ILP) named *Unlimited Platooning Problem - Different Starting Nodes*, the vehicles can have various starting nodes. In this formulation, vehicles can traverse over the same edge at different times and consequently in different platoons.

Moreover, [57] introduces a best-pair and hub heuristics to solve the vehicle platooning problem more efficiently. The best-pair algorithm compares pairwise all vehicle routes to find an optimal vehicle pair and their merging and splitting nodes. When a vehicle pair is found, it will be reduced to a one-vehicle route with merging node as start and splitting node as the destination. The algorithm repeats the comparison until no further platoons can be formed. The complexity of the *Best Pair* heuristic is $\mathcal{O}(N^2 \log N \cdot |V|^2)$, where the N is a number of vehicle assignments, and V is a number of road points in the road network.

The idea of a *Hub Heuristic* is to split vehicles into platoons, select a certain node (hub) for each platoon and solve the same-start VPP from hub to start node and from hub to a destination node. The heuristic rates all edges for each vehicle (edge vector). The ratio between the vehicle shortest path and the vehicle shortest path over a selected edge determines a probability that this edge will be used by a vehicle. Initially, each vehicle is converted into individual platoon, then the algorithm compares pairwise the platoons and merges the strongest pairs into single platoons. The compatibility of two platoons can be determined by element-wise multiplication of the edge vectors. The sum of the vector elements represents compatibility.

The vehicles routing from their start nodes to a hub node and from the hub node to their end node can be solved with a best-pair heuristic. The complexity of the *Hub Heuristic* is $\mathcal{O}(N^2 \log N \cdot |E|)$, where the N is a number of vehicle assignments, and V is a number of road points in road network.

For the experiments, [57] uses the German road network with 647 nodes and 1390 edges. For the same-start problem instances, up to 200 vehicles were used. More difficult problem instances with different starting nodes and up to ten vehicles have also been tested. Solving the unlimited platooning problem instance with ten vehicles achieves an average fuel savings of 1.5%. The *Best Pair* heuristic achieves near-optimal results. The fuel savings of the *Hub Heuristic* are uniformly distributed between 0% and 1.5%. The presented heuristics are only tested on small graphs with a small number of vehicles. The complexity of both algorithms depends on the graph size and the number of vehicles. For large graph instances, the performance does not scale well.

[55, 86] propose an optimization model that can determine the fuel-optimal platoon routings and waiting times. The experiments were performed on a 10×10 grid road network and a road network with 4553 nodes. [74] introduces the non-linear optimization problem with time constraints for the platooning problem. A novel genetic algorithm is proposed to solve the problem heuristically. For the experiments, a German road network with 20 essential cities and a valency of 2 for the other nodes are used.

[72] proposes an ant colony optimization algorithm to solve the platooning problem. This work uses six different road networks with a maximum of 500 nodes.

[73] proposes a linear program formulation for the platooning problem, which takes the different vehicle speeds and time constraints into account. The presented algorithm uses a swarm intelligence concept to solve problem instances with up to 1000 vehicles on the Chicago road network.

3.2 Related Problems

This section reviews the related problems and discusses the proposed approaches.

3.2.1 Trip Grouping Approaches

[80] investigates the taxi passengers' assignment into groups with optimal ride incentive, based on New York taxi trip data. The taxi trip data for each passenger consists of a trip origin and destination location, origin, and destination timestamp, trip cost and trip length. To determine optimal passenger groups for ridesharing, [80] introduces the *Group Ride Graph* (GRG). Passengers are defined as nodes, and edges represent the ability and incentive to share a ride for two passengers, i and j. Two passengers, i and j, are only combinable if they benefit from the group ride. Thereby, extra walking miles and additional waiting time are considered. The driver gets additional revenue for participating in ridesharing, which depends on the number of passengers. Cliques in the *Group Ride Graph* (GRG) are defined as stable ride groups. Based on these groups, the authors have implemented *Induced Group Ride Graph* (IGRG), where nodes defined as stable groups. The groups with identical passengers are related to each other. This work's major contribution is to assign passengers to taxis with maximum fuel-saving, based on *Group Ride Graph*

(GRG). There were three algorithms proposed to solve this problem. The exact algorithm initially determines in IGRG connected components and their weights. Furthermore, the algorithm branches each component recursively into subcomponents by selecting nodes with the greatest degree. Finally, the algorithm backtracks the weight from leaf to root and chooses a maximum weight path. The exact algorithm does not scale well for a large number of passengers. There were *heuristic* and *greedy* algorithms introduced to assign passengers to near-optimal groups efficiently. The heuristic approach uses spectral bisection to partition the GRG in two subgraphs with an equal number of nodes. The exact algorithm is used to find a solution in separated subgraphs. The greedy algorithm finds disjoint sets of passengers with maximum weight more efficient. However, the results show a robust negative deviation from the exact results.

Twentry experiments were conducted with a small number of passengers, in a range from 41 to 244. The road network size and complexity were not considered in experiments, just urban area was expected. Only stable groups of passengers were used to form rider groups. This limitation can disqualify a lot of reasonable solutions. Moreover, passengers can only share a ride if their origin and destination locations are in the same regions. This assumption prunes also a lot of profitable rides, e.g., passengers cannot join a ride on the midway of another passenger.

The approach in [51, 48] groups the trips based on their similarity. The trips are vectors in three-dimensional Euclidean space, where the z-axis represents the time. The technique helps to optimize the routes and scheduling process of demand-responsive transportation.

3.2.2 Column Generation Approaches

[23] presents a column generation approach for the Vehicle Routing Problem with Time Windows. The VRPTW is formulated as a set partitioning problem, selecting the cost-minimum set of routes, taking into account the corresponding constraints. The columns in these formulations represent the feasible routes; a vast number of columns require the column generation method.

[100] uses the column generation approach to route the homogenous Unmanned Aerial Vehicles (UAV). The problem is modeled as a variant of the Vehicle Routing Problem (VRP) with multiple target and pickup points. As usual, the master problem is based on a path formulation.

Chapter 4

Research Goal and Contributions

4.1 Research Goal and Objectives

The overall research goal of this thesis is to propose and evaluate novel efficient and effective algorithms for the fuel-optimal vehicle platooning problem that cope well with large-scale data-intensive problem instances. Our research is mainly attributed to the field of centralized platooning. More specifically, the focus will be on: (1) the development of novel strategies for platooning-driven vehicle grouping, which includes the analysis of pairwise and groupwise platooning incentives and the formation of vehicle groups for platooning, (2) the development of novel efficient heuristics for independent near-optimal platoon routing of each of the promising vehicle groups, and (3) the development of a novel graph data model to provide native graph database support for the management, processing, and querying platooning relevant data.

Our research aims to develop vehicle grouping methods combined with independent vehicle routing heuristics for effectively and efficiently handling large-scale data-intensive instances of the vehicle platooning problem. **The research goal described above will be achieved by completing the following set of objectives.**

(1) To develop novel strategies for detecting promising vehicle groups for forming vehicle platoons based on their pairwise and groupwise fuel-saving potential that can be effectively estimated without the expensive computation of entire routings, and to propose fast and scalable grouping algorithms that implement these strategies.

(2) To develop novel efficient algorithms for finding near-optimal routings of promising vehicle groups that can be applied independently of other vehicles, and to propose scalable combinations of grouping methods and routing methods, thus enabling the division of large-scale data-intensive problem instances into tractable subproblems.

(3) To develop a novel property graph data model to store and process platooning relevant data, such as road networks, vehicle assignments, vehicles paths, and platooning incentives, natively in a graph database to unlock the capabilities of modern graph database management systems and develop efficient algorithms that make effective use of these capabilities to cope with large-scale data-intensive problem instances.

4.2 Research Contributions

Below we list the major contributions of this thesis to the state-of-the-art in fuel-optimal vehicle platooning algorithms. With respect to Objective (1), our major contributions are the following:

- We define pairwise and groupwise platooning incentives and propose methods for calculating them. Our methods work independently from the road network size so that the calculation can be done efficiently, even for large numbers of vehicles. We propose three lightweight incentive methods, which are based on inexpensive geometric operations. The most effective method constructs a geometric container for each vehicle and compares the respective containers with each other.

- Based on our platooning incentives, we propose grouping algorithms to determine vehicle groups with high potential for platooning. This approach significantly reduces the computation time for platoon formation and routing. We discuss different grouping strategies and establish a relationship with finding clique partitions in graphs. Using this relationship, we provide algorithms (α-*cut Method*, *Greedy CPP*, *SGVNS CPP*) to realize our grouping strategies and compare them to their capability to form near-optimal vehicle groups for fuel-optimal platooning.

With respect to Objective (2), our major contributions are the following:

- We propose a Column Generation model for vehicle platooning. Based on the existing integer linear programming model for vehicle platooning, we give an equivalent path-based formulation, i.e., master problem. Using the Column Generation technique, we propose a new algorithm that restricts and solves the master problem iteratively. To generate new promising platooning paths for the restricted master problem, we formulate a subproblem. This approach reduces the computation time for finding near-optimal solutions considerably.

- We examine the combination of our proposed vehicle grouping and vehicle routing algorithms. Our experimental evaluation shows that using our combined method makes it possible to **solve large-scale problem instances with up to 300,000 road segments and 2000 vehicles**. This is a significant improvement over the current state-of-the-art.

- We propose a time scheduling algorithm for predetermined platooning routings. Our algorithm provides vehicle travel times and scheduling with considerations of the time constraints.

With respect to Objective (3), our major contributions are the following:

- We propose a labeled graph property model for vehicle platooning by analyzing the essential tasks to be performed in our algorithms and the corresponding data flow. Our model represents all platooning relevant data such as the road network, vehicles and vehicle assignments, platooning incentives, vehicle groups, shortest paths, platooning paths, and the corresponding index structures. To further optimize the query performance and speed up our algorithms, our model is based on the filter-based modeling rule, see [89].

- For the experimental evaluation of our algorithms, we extract the data of real-world road networks from publicly available sources, i.e., OSM, transform it to our property graph data model and import it into a graph database that implements our property graph data model, as shown in [90].

- We develop a well-architectured prototype, which incorporates our graph database and implements our proposed algorithms. The modular structure of the prototype enables the flexible combination and extension of our algorithms. The

system provides an infrastructure for storing, retrieving, and caching mission-critical data. The test cases of our experiments can be easily parameterized, executed, and analyzed when using our prototype.

4.3 Limitations

Similar to most existing works on fuel-optimal vehicle platooning [57], we assume that vehicles travel at a constant speed with constant fuel consumption. Some works deal with fuel-optimal speed adjustments for existing platooning routes or decentralized platooning, as seen in [92, 93, 54, 59, 4, 60].

In this thesis, we assume a 10% fuel reduction rate for vehicles driving in a platoon, except for the leading vehicle in the platoon. Fuel reduction rates have been studied in the literature with varying outcomes, such as in [11, 7].

In this thesis, we do not assume time restrictions such as the latest departure time or earliest arrival time, as done in some other works, such as in [57]. Our proposed platooning incentives can be adapted for the problem with time restrictions, considering an additional dimension for the time. Based on such incentives, vehicle groups can be formed using grouping algorithms. Time constraints can also extend our formulation of the Column Generation approach. However, this is out of the scope of this thesis.

Chapter 5

Heuristics for Finding Near-Optimal Groups for Vehicle Platooning

To approach VPP, we divide it into two phases. In the first phase, the vehicles are assigned into near-optimal groups with high fuel saving potential. This is the subject of the current chapter. In the second phase, we compute optimal or near-optimal routes for platooning, based on the groups found in the first phase. This will be discussed in Chapter 6.

This chapter proposes a fast heuristic to find optimal groups for a large number of vehicles in large road networks. We have already published the initial idea of this approach in [88]. This chapter extends this approach. Algorithm 1 provides the abstract steps of the grouping-based routing algorithm. Section 5.1 introduces the concept of pairwise platooning incentives. These are computed using several lightweight methods to analyze the platooning opportunities for each vehicle pair. Section 5.2 introduces approaches to form near-optimal groups based on platooning incentives. Section 5.3 deals with the dissolution of the non-disjoint groups, without expensive computation of platooning routing. Steps 4 and 5 will be discussed in Chapter 6.

Algorithm 1 General Grouping-based Routing Algorithm

Data: Road network $G = (V, E)$ and vehicle set H.
Result: (Optimal) platoon routing $O = \{P_1^{pl}, \ldots, P_{|H|}^{pl}\}$ for vehicle set H.

1: **Computation of Pairwise Incentives:** *Algorithm 2* (PIH) (Section 5.1)
2: **Formation of Candidate Groups:** (Section 5.2)
   Grouping Algorithm $\in \{\alpha$*-cut Method, Exact CPP, Greedy CPP, SGVNS CPP*$\}$
3: **Resolution of Non-Disjoint Groups:** (Section 5.3)
4: **Computation of near-optimal Platoon Routing:** (Section 6)
   Routing Algorithm $\in \{$*Exact Solver, Column Generation*$\}$
5: **Selection of the Best Mutually Disjoint Groups:** (Section 6.4)

5.1 Computation of Pairwise Incentives

This section proposes a fast heuristic to determine the routes similarities or dissimilarities for each vehicle pair in set H. The pairwise comparison is a powerful method to determine relative similarity between all vehicle routes [48]. The number of pair comparisons is $\frac{|H|^2-|H|}{2}$ for the $|H|$ vehicles. The similarity of two vehicle routes

with regard to platooning captures how beneficial it is for these two vehicles to form a platoon and achieve fuel savings. The basic idea of the proposed heuristic is to identify the platooning opportunities of vehicle pairs without expensive computation of platooning routes. The directions and spatial proximity of the considered vehicles are particularly important.

Algorithm 2 introduces our Pairwise Incentive Heuristic (PIH), which computes pairwise incentives for a given set of vehicles. Our heuristic consists of three methods; these are described in Section 5.1.1, Section 5.1.2 and Section 5.1.3. The methods are called sequentially, see Lines 3 - 5 and computed incentives are stored in the matrix R. For simplification, the methods are aggregated into a single method $\lambda : H \times H \mapsto [0, 1]$. The particular methods can be weighted by parameters $\varepsilon_1, \varepsilon_2$ and ε_3, see Eq. 5.2. The vehicle pair element ϱ is introduced in Eq. 5.1 and is used in further formulas.

$$\varrho \in \{(h, p) \in H \times H : h \neq p\} \tag{5.1}$$

$$\lambda : \varrho \mapsto \lambda_1(\varrho) \cdot \varepsilon_1 + \lambda_2(\varrho) \cdot \varepsilon_2 + \lambda_3(\varrho) \cdot \varepsilon_3 \tag{5.2}$$

$$\text{with} \quad \varepsilon_1 + \varepsilon_2 + \varepsilon_3 = 1 \quad \varepsilon_1, \varepsilon_2, \varepsilon_3 \in [0, 1] \tag{5.3}$$

Algorithm 2 Pairwise Incentive Heuristic (PIH)

Data: Road network $G = (V, E)$, vehicle set H and weights $\varepsilon_1, \varepsilon_2$ and ε_3.
Result: Incentive matrix R.

1: **for** $i = 1 \ldots |H| - 1$ **do**
2: **for** $j = i + 1 \ldots |H|$ **do**
3: $R_{i,j} = \lambda_1(\textit{angle between direction vectors of vehicles } h_i \textit{ and } h_j) \cdot \varepsilon_1$ (sec. 5.1.1)
4: $R_{i,j} = R_{i,j} + \lambda_2(\textit{vectors of vehicles } h_i \textit{ and } h_j) \cdot \varepsilon_2$ (sec. 5.1.2)
5: $R_{i,j} = R_{i,j} + \lambda_3(\textit{shortest paths of vehicles } h_i \textit{ and } h_j) \cdot \varepsilon_3$ (sec. 5.1.3)

In Eq. 5.4, the platooning incentive matrix R is defined as a lower triangular matrix, which includes λ incentive values for all vehicle pairs. In Section 5.2, the matrix R will be used to form candidate vehicle groups with high fuel saving potential.

$$R = \begin{pmatrix} 0 & 0 & \ldots & 0 \\ \lambda(h_1, h_2) & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \lambda(h_1, h_{|H|}) & \ldots & \lambda(h_{|H|-1}, h_{|H|}) & 0 \end{pmatrix} \tag{5.4}$$

5.1.1 Pairwise Comparison of Vehicle Direction Vectors

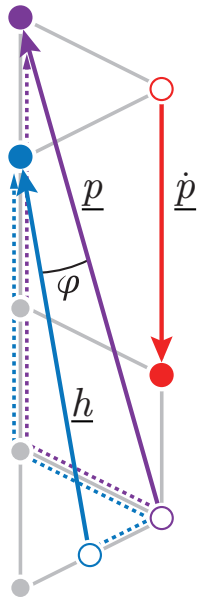
This approach compares the directions of two vehicle assignments in order to reject vehicle pairs that are unusual for platoon routing. In other words, vehicles which travel in different directions are not suitable to form a platoon. In this case, no further checks are necessary for the given vehicle pair. The vehicle direction vector $\underline{h}$, see Eq. 5.5, is computed from the start node a and the end node b of the vehicle assignment h. Fig. 5.1 illustrates the vehicle vectors.

$$\underline{h} = \begin{pmatrix} \gamma(b) - \gamma(a) \end{pmatrix} \tag{5.5}$$

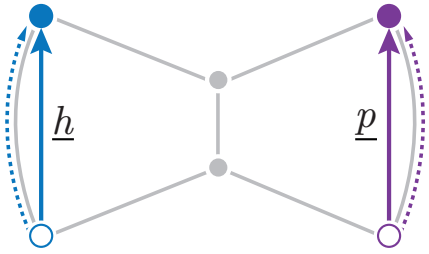
The degree of similarity between two vehicle routes with regard to platooning can be partially determined by the angle between the vehicle vectors $\underline{h}$ and $\underline{p}$. A small angle may indicate a "good" platooning opportunity, while a large angle may indicate a "bad" platooning opportunity. This condition can be checked very efficiently. Eq. 5.6 shows the formula of the angle φ between the vectors $\underline{h}$ and $\underline{p}$.

$$\varphi = cos^{-1}\left(\frac{\underline{h}\cdot\underline{p}}{|\underline{h}|\cdot|\underline{p}|}\right) \tag{5.6}$$

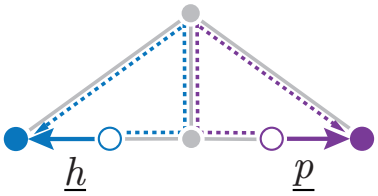
The basic idea is illustrated in Fig. 5.1a. This example shows the direction vectors of vehicles h (blue) and p (violet) on a gray colored road network, with relatively small angle $\varphi \approx 6°$ between the vectors. In this case, vehicles h and p happen to be "good" candidates for a platooning Fig. 5.1a, because most parts of the shortest paths of the two vehicles (dotted lines) are nearby or even overlapping. Vehicle $\dot{p}$ with red vector $\underline{\dot{p}}$ travels in the opposite direction of the vehicle p with large angle $\measuredangle(\underline{p},\underline{\dot{p}}) \approx 164°$ among them; for this reason, the vehicles $\underline{p}$ and $\underline{\dot{p}}$ cannot platoon together, see Fig. 5.1a. A necessary condition for the platooning of two vehicles is that the angle between their vectors is less than $90°$.



(a) The angle between $\underline{h}$ and $\underline{p}$ vectors is relatively small with $\varphi \approx 6°$. A priori the shortest paths of vehicles h and p have a high similarity.



(b) The angle between $\underline{h}$ and $\underline{p}$ vehicle direction vectors is $\varphi = 0°$; however, vehicles h and p cannot form a platoon.



(c) The vehicle direction vectors $\underline{h}$ and $\underline{p}$ are directed in opposite directions although the vehicles h and p can form a platoon.

FIGURE 5.1: Examples of pairwise comparison of vehicle directions.

$$\lambda_1 : \mathbb{R} \mapsto [0,1] \tag{5.7}$$

$$\lambda_1 : \varphi \mapsto \begin{cases} cos(\varphi) & \text{if } \varphi \leq 90° \\ 0 & \text{if } \varphi > 90° \end{cases} \tag{5.8}$$

Eq. 5.8 defines a function λ_1 that determines a value between 0 and 1 for each angle φ. For example, a vehicle pair with $\varphi \approx 90°$ is not suitable for platooning and the corresponding function value is $\lambda_1(90°) = 0$. This approach rejects unsuitable

vehicle pairs from the grouping process very efficiently, which is introduced in Section 5.2. The pairwise comparison of the vehicle directions does not provide enough information to form "good" candidate groups with high platooning potential. However, it helps to identify unsuitable vehicle pairs. Fig. 5.1b shows an example with two vehicles, traveling in the same direction; that is, with $\varphi = 0°$. The heuristic indicates high platooning potential because $\lambda_1(0°) = 1$. Nevertheless, the given vehicles cannot form a fuel-efficient platoon, since the distance between the vehicle vectors is large. Fig. 5.1c shows another special case with a large angle $\varphi = 180°$ between the two vehicle vectors. In this case, the method provides an incentive with a value 0.

5.1.2 Spatial Proximity of Two Vehicle Assignments

This section introduces the second incentive method, see line 4. It provides a more precise assessment about route similarity in terms of platooning. The idea is to solve the vehicle platooning problem in Euclidean space, instead of a road network G, to find more routes similarities for given vehicles. The vehicle platooning problem in Euclidean space is defined in Def. 5.1.1.

Definition 5.1.1 (Euclidean platooning problem)**.** The Euclidean platooning problem consists of finding an optimal platoon routing for a finite set of vehicle assignments in 2-dimensional Euclidean space.

An example of the Euclidean platooning problem is shown in Fig. 5.2. The sets of vectors $\{(a_h, z_1)^\intercal, (z_1, z_2)^\intercal, (z_2, b_h)^\intercal\}$ and $\{(a_h, z_1)^\intercal, (z_1, z_2)^\intercal, (z_2, b_h)^\intercal\}$ represent platoon routings for vehicles h and p in the 2-dimensional Euclidean space, respectively. Vector $(z_1, z_2)^\intercal$ is the common part of the platooning routes, with merge point z_1 and split point z_2. The traveling costs on vector $(z_1, z_2)^\intercal$ are reduced by the saving factor η' for one vehicle. The approach finds optimal points z_1 and z_2 for vehicles h and p and consequently the linear platooning routing.

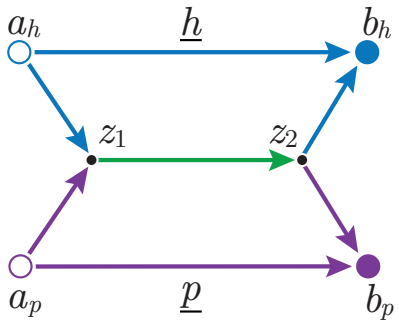


FIGURE 5.2: An example of the Euclidean platooning problem for vehicles h and p.

Eq. 5.9 defines a convex distance function with parameters $z_1, z_2 \in \mathbb{R}^2$ and with fixed start and end coordinates of two vehicles. The global minimum of the function provides a fuel-optimal platooning distance in the 2-dimensional Euclidean space for a vehicle pair, according to the η' factor. The Euclidean platooning problem, can be solved with the proposed function in Eq. 5.9. It should be noted that the Euclidean platooning problem is similar to the Steiner tree problem with two Steiner points, see also the geometric median [25, 98].

$$g : (z_1, z_2) \mapsto \quad d(\gamma(a_h), z_1) + d(\gamma(a_p), z_1) + d(z_2, \gamma(b_h)) + d(z_2, \gamma(b_p)) + d(z_2, z_1) + d(z_2, z_1) \cdot (1 - \eta') \tag{5.9}$$

Eq. 5.10 illustrates the difference between the shortest linear distance and the Euclidean platooning distance, see Eq. 5.9, of the vehicles h and p. When the linear distance is shorter than the Euclidean platooning distance, then the vehicles h and p are not suitable for platooning on the road network. The Euclidean norm is represented by $|\cdot|$.

$$|\underline{h}| + |\underline{p}| \leq \min_{z_1, z_2 \in \mathbb{R}^2} g(z_1, z_2) \tag{5.10}$$

The optimization problem $\min_{z_1, z_2 \in \mathbb{R}^2} g(z_1, z_2)$ needs to be solved to check the condition Eq. 5.10. The function g is a distance function and therefore a strictly convex function with a unique global minimum point. Hence, the optimal solution can be found with different local search methods [27, 6, 20] efficiently.

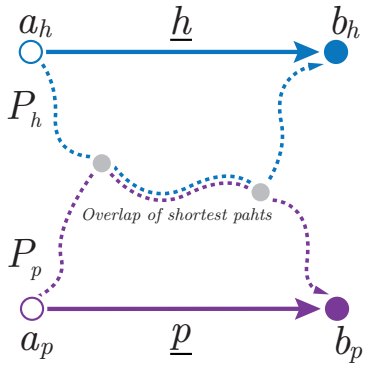


FIGURE 5.3: The fuel-efficient linear platoon with $\eta' = 0.1$ cannot be formed although the shortest paths of vehicles h and p overlap.

The assumption made in the heuristic strategy can cause many problems when we look at road networks. The Fig. 5.3 shows an example with two vehicles, in which no fuel-efficient linear platooning is possible; Due to the distance between the direction vectors of the vehicles is too large. However, the shortest paths of the vehicles h and p have common road segments in the given example, in which a fuel-efficient platoon can be formed. The example previously introduced in Fig. 5.1c will also be rejected by the condition Eq. 5.10. To address such cases, the linear savings factor η' should be chosen flexibly, depending on the road network, transport assignments and traffic congestion. By default, the saving factor η' is set to 0.5, which increases the tolerance for the condition Eq. 5.10.

The previously mentioned cases will not be pruned and can be checked in the further steps. Fig. 5.4 shows an opposite example with a small gap between the direction vectors. Here, the heuristic indicates high platooning potential with large savings in the Euclidean space. Due to a river, there are no roads between the direction vectors and therefore no platooning is possible.

Our heuristics take into account the combination of direction and distance of vehicle vectors. This makes it possible to reject some special cases such as the case in Fig. 5.1b. This method specifies also a meaningful similarity index based on direction vectors only. The similarity index of a vehicle pair can be determined by the function, which is specified in Eq. 5.11. The function has as input the direction

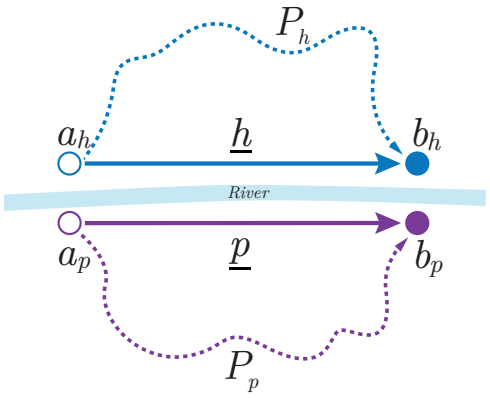


FIGURE 5.4: The heuristic indicates high platooning potential; however, no platooning is possible in this example.

vectors $\underline{h}$ and $\underline{p}$ and as output a value in the interval $[0, 1]$. If the condition defined previously (Eq. 5.10) is true, then the result of the function is 0 and it means, platooning is not possible. The rest of the function specifies a normalized ratio $]0, 1]$ between the length of direction vectors and the linear platooning routing.

$$\lambda_2 : (\underline{h}, \underline{p}) \mapsto \begin{cases} \left(\frac{|\underline{h}|+|\underline{p}|}{\min\limits_{z_1,z_2 \in \mathbb{R}^2} g(z_1,z_2)} - 1 \right) \cdot \left(\frac{2}{\eta'} - 1 \right) & \text{if } |\underline{h}| + |\underline{p}| > \min\limits_{z_1,z_2 \in \mathbb{R}^2} g(z_1, z_2) \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

5.1.3 Pairwise Comparison of Vehicle Geometric Containers

The concept of Vehicle Geometric Convex Container (VGCC) is introduced in Definition 5.1.2, which can enormously reduce the search space in graphs or road networks [96]. The maximum detour of each vehicle is limited by the saving factor η. Fig. 5.5 shows two different types of Vehicle Geometric Containers $\Gamma_L(P_h)$ and $\Gamma_K(P_h)$.

Definition 5.1.2 (Vehicle Geometric Convex Container (VGCC))**.** A Vehicle Geometric Convex Container $\Gamma(P_h)$ is an abstract construct, which is defined as a convex subset in $\mathbb{R}^2$. The VGCC covers nodes with feasible detour distances from the shortest path P_h in terms of platooning.

Further containers can be derived from the abstract VGCC, e.g. $\Gamma_L(P_h)$ or $\Gamma_K(P_h)$ containers, see below. The Vehicle Geometric Convex Container $\Gamma_L(P_h)$ is an ellipse, which represents an upper bound area for a feasible detour from the shortest path. Definition 2.1.2 already introduces the upper bound costs of a platooning path P_h^{pl}. All such paths are covered by the ellipse. The linear distance between start a and end b node via a 2-dimensional point Θ is less than or equal to $c(P_h) \cdot \frac{1}{1-\eta}$. The formal description is illustrated in Eq. 5.12. An additional parameter $\beta \in (0, 1]$ with a default value of 1, makes it possible to control the size of the ellipse. The VGCC $\Gamma_L(P_h)$ contains all feasible solutions and can reduce the road network enormously. On the other hand, the ellipse overestimates the searching area.

$$\Gamma_L : P_h \mapsto \left\{ \Theta \in \mathbb{R}^2 : \|\Theta - \gamma(a_h)\| + \|\Theta - \gamma(b_h)\| \leq c(P_h) \cdot \left(\frac{1}{1-\eta} \right) \cdot \beta \right\} \tag{5.12}$$

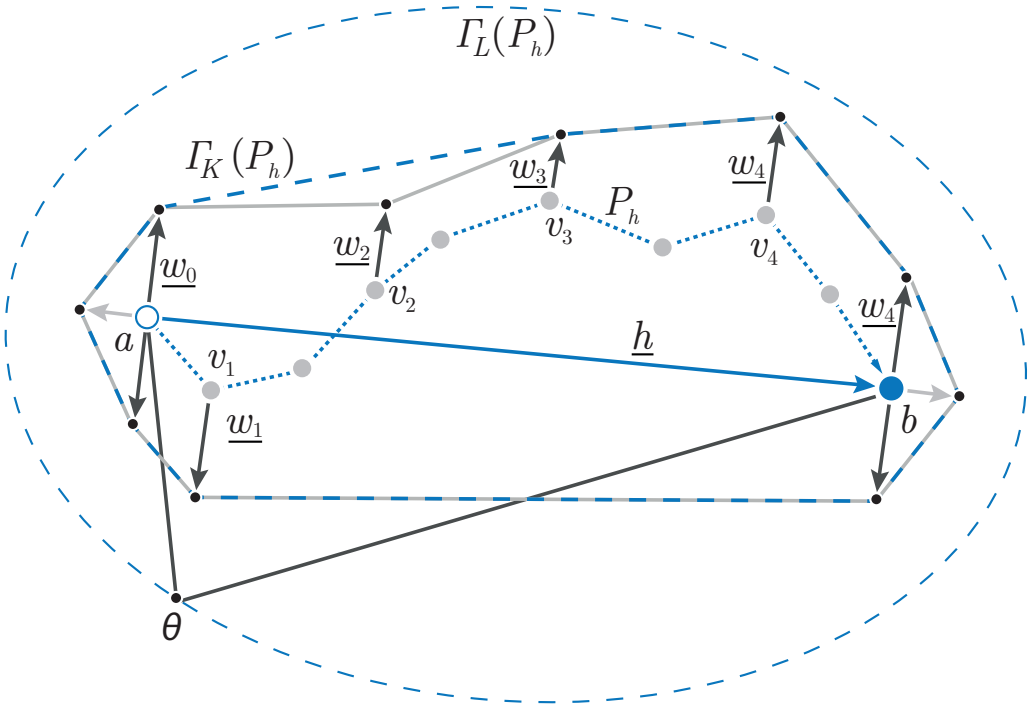


FIGURE 5.5: Example construction of the Vehicle Geometric Containers $\Gamma_L(P_h)$ and $\Gamma_K(P_h)$.

The Vehicle Geometric Convex Container $\Gamma_K(P_h)$ is a polygon with $\Gamma_K(P_h) \subseteq \Gamma_L(P_h)$. To construct $\Gamma_K(P_h)$ the approach selects particular nodes from the shortest path P_h. For each selected node we compute $\underline{w}$ vectors, which are orthonormal to the vector h. The length of the initial vector $\underline{w}_0$ is a maximal possible retour of the shortest path P_h Eq. 5.13 shows the detailed definition of the vector $\underline{w}_0$. The function f_K in Eq. 5.14 determines a vector for the selected nodes $v \in P_h$, based on the initial vector $\underline{w}_0$. The $\underline{w}$ vectors are the smallest in the middle of the path and the vectors become larger towards the outside. The endpoints of the vectors are used to determine the convex hull, which gives the Vehicle Geometric Convex Container $\Gamma_K(P_h)$.

$$\underline{w}_0 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \frac{\underline{h}}{|\underline{h}|} \cdot c(P_h) \cdot \left(\frac{1}{1-\eta} - 1 \right) \tag{5.13}$$

$$f_K : v \mapsto \begin{cases} |\underline{w}_0| \cdot \left(1 - \frac{c(P_h^{a,v})}{c(P_h)}\right) & \text{if} \quad \frac{c(P_h^{a,v})}{c(P_h)} \leq \frac{1}{2} \\ |\underline{w}_0| \cdot \frac{c(P_h^{a,v})}{c(P_h)} & \text{else} \end{cases} \qquad v \in P_h \tag{5.14}$$

The idea of this approach is to cover most paths with high platooning potential for a given vehicle and to reduce more searching space than the ellipse based approach. The computation in further steps is very expensive. This approach tries to determine the best tradeoff between accurate results and execution performance.

Definition 5.1.3 introduces the VGCC Intersection concept, which is used as a basis for the pairwise comparison. Fig. 5.6 is an illustrative example of the VGCC Intersection (red line) of Vehicle Geometric Containers $\Gamma_L(P_h)$ and $\Gamma_L(P_p)$. The platooning routes are shown by green dashed lines. The common part of platooning route starts at v_m node and ends at v_s node and it locates inside of a VGCC Intersection. The common part of platooning path cannot be outside of the intersection area $\Gamma_L(P_h) \cap \Gamma_L(P_p)$. The red path within an intersection area cannot

be used by vehicle h, due to a too large detour; this is an example of overestimation of the searching area.

Definition 5.1.3 (VGCC Intersection)**.** The intersection of at least two Vehicle Geometric Containers produces a single convex subset of the intersected VGCC or an empty set $\emptyset$. In the non-empty intersection area, the vehicles can platoon together. In other words, if the Vehicle Geometric Containers do not intersect, then the corresponding vehicles will no longer be considered.

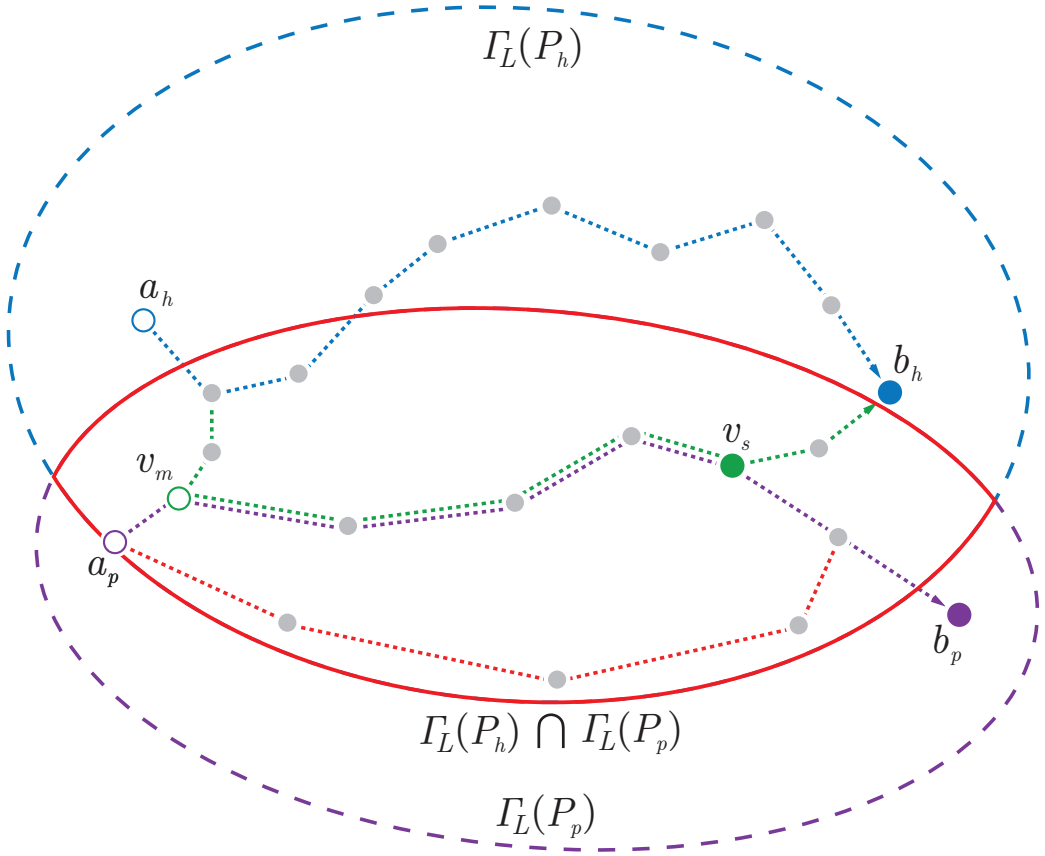


FIGURE 5.6: The VGCC Intersection of $\Gamma_L(P_h)$ and $\Gamma_L(P_p)$, with shared path(green) of h and p.

Fig. 5.7 depicts a VGCC Intersection of Vehicle Geometric Containers $\Gamma_K(P_h)$ and $\Gamma_K(P_p)$ this area is much smaller than the intersection area, in Fig. 5.6. The size of VGCC Intersection influences directly the number of the groups and performance of the route computations in each group; the context of which is explained in Section 5.2. The overestimated red path in Fig. 5.6 is correctly filtered out from further computations by $\Gamma_K(P_h) \cap \Gamma_K(P_p)$. The platooning path is filtered out on the downside as well.

Fig. 5.8 depicts a Vehicle Geometric Convex Container $\Gamma_{K^+}(P_h)$, which is an extended version of the Vehicle Geometric Convex Container $\Gamma_K(P_h)$. The VGCC $\Gamma_{K^+}(P_h)$ is a VGCC $\Gamma_K(P_h)$ combined with VGCC $\Gamma_{K'}(P_h)$, which is mirrored at vector $\underline{h}$. The Fig. 5.8 shows the shortest path P_h and the alternative path (red) with slightly higher costs. The alternative path is not covered by VGCC $\Gamma_K(P_h)$ although it has high platooning potential. VGCC $\Gamma_{K^+}(P_h)$ borrows symmetry properties from VGCC $\Gamma_L(P_h)$ and maintains a relatively small area.

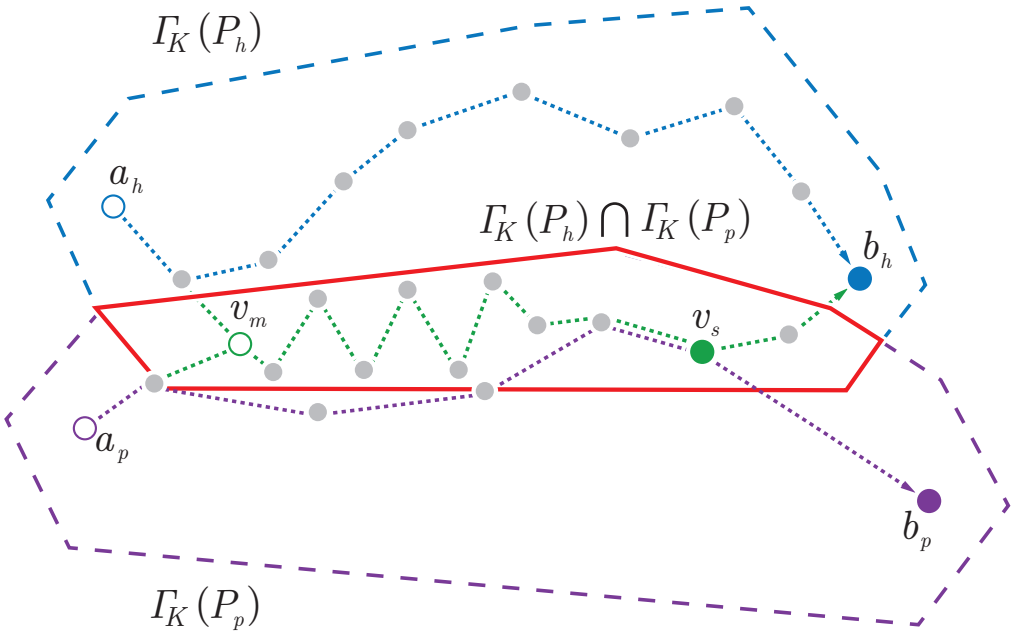


FIGURE 5.7: The VGCC Intersection of Vehicle Geometric Containers $\Gamma_K(P_h)$ and $\Gamma_K(P_p)$.

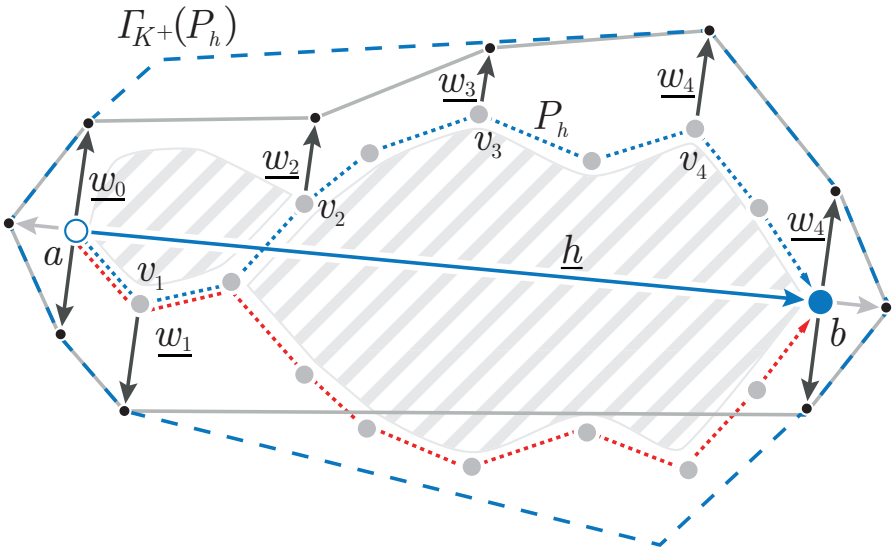


FIGURE 5.8: The Vehicle Geometric Convex Container $\Gamma_{K^+}(P_h)$, an extended version of Vehicle Geometric Convex Container $\Gamma_K(P_h)$.

Eq. 5.15 shows incentive function λ_3, which returns a value between 0 and 1, based on the intersected area of the shortest paths P_h and P_p. The intersection area $A(\Gamma(P_h) \cap \Gamma(P_p))$ divided by the VGCC area $A(\Gamma(P_h))$, gives the spatial proximity between vehicles h and p relative to vehicle h. The incentive function λ_3 provides an intersection area, normalized relative to the original containers.

If the shortest paths P_h and P_p are exactly the same, then the function value between vehicles h and p is 1. The intersection area completely covers the area of containers $\Gamma(P_h)$ and $\Gamma(P_p)$; consequently, the vehicles h and p will form a platoon. In the event of an empty intersection area, the function value is 0 and the vehicles h and p cannot form a platoon.

$$\lambda_3 : (P_h, P_p) \mapsto \frac{1}{2} \cdot \left(\frac{A(\Gamma(P_h) \cap \Gamma(P_p))}{A(\Gamma(P_h))} + \frac{A(\Gamma(P_h) \cap \Gamma(P_p))}{A(\Gamma(P_p))} \right) \qquad \Gamma(P_h), \Gamma(P_p) \neq \emptyset \tag{5.15}$$

5.2 Formation of Candidate Groups

In the previous Section 5.1 the incentive function λ is introduced, which determines an incentive value for each vehicle pair $H \times H$. This section introduces concepts of the vehicle grouping based on incentive values. The Definition 5.2.1 specifies the term vehicle group [33].

Definition 5.2.1 (Vehicle Group $\tilde{C}$)**.** The vehicle group (or group for short) is a subset of vehicles $\tilde{C} \subseteq H$ with pairwise incentives greater than or equal to alpha-cut value $\tilde{\alpha}$ [48]. The alpha-cut value $\tilde{\alpha}$ can be selected from the interval $(0, 1]$. Each group satisfies the conditions in Eq. 5.16 and contains at least two vehicles.

$$\forall h, p \in \tilde{C} : \lambda(h, p) \geq \tilde{\alpha} \wedge h \neq p \tag{5.16}$$

The number, size of groups, similarity of grouped vehicles and thus also the platooning potential can be controlled with the alpha-cut value $\tilde{\alpha}$. A group includes vehicles with potential to form platoon(s) (Def. 2.1.1).

The function $\tilde{w}$, in Eq. 5.17, cumulates incentive values of all unordered pairs from vehicle group $\tilde{C}$. This function determines group incentive, which is a measure of the xgroup quality. The weighting function $\tilde{w}$ is very often used by graph clustering and partitioning algorithms [13]. Therefore, it is here used as a cumulative measurement to indicate the savings potentials of a group.

$$\tilde{w} : \tilde{C} \mapsto \frac{1}{2} \cdot \sum_{\substack{h,p \in \tilde{C}: \\ h \neq p}} \lambda(h, p) \tag{5.17}$$

$$\tilde{w}' : \tilde{C} \mapsto \frac{1}{2 \cdot \binom{|\tilde{C}|}{2}} \cdot \sum_{\substack{h,p \in \tilde{C}: \\ h \neq p}} \lambda(h, p) \tag{5.18}$$

The general strategy of the heuristic is to form disjoint vehicle groups with maximized overall incentives. The disjoint vehicle groups are defined in (Def. 5.2.3). The group incentives can be calculated in different ways; Eq. 5.17 is a standard group quality function. Eq. 5.18 describes an alternative quality function, which determines the average of the pair incentives for the group $\tilde{C}$.

Definition 5.2.2 (Vehicle Group Set)**.** The formal definition of group set is defined in Eq. 5.19.

$$\tilde{S} = \{\tilde{C}_1, \ldots, \tilde{C}_n\} \;\; n \in \mathbb{N}, \tilde{C}_1, \ldots, \tilde{C}_n \;\; \text{groups} \tag{5.19}$$

Definition 5.2.3 (Disjoint Vehicle Group Set)**.** The vehicle group set $\tilde{S}$ is pairwise disjoint if $\forall\ \tilde{C}_i, \tilde{C}_j \in \tilde{S} : i \neq j \Rightarrow \tilde{C}_i \cap \tilde{C}_j = \emptyset$. The number of pairwise disjoint vehicles groups is limited by $\frac{|H|}{2}$.

A groupwise calculation of platooning routes reduces the complexity enormously. In some cases it is reasonable to assign the same vehicle into multiple groups. The group incentives give only an approximation of the group quality caused by a varying ratio between group incentives and group savings. The groupwise computation of platooning routes and corresponding savings (Section 6) are the most computational expensive part. The number of disjoint groups is limited by the number of vehicles. The group with maximum incentives is not necessarily the group with the most savings, because the consideration of only disjoint groups can prune good solutions. The number of non-disjoint groups can be much larger than the number of disjoint groups, and the computation of savings for a huge number of groups is still inefficient. The disjoint group set can be extended by reasonable groups to control the tradeoff between results quality and computation time.

5.2.1 Alpha-Cut Grouping Method (α-*cut Method*)

Algorithm 3 determines the set of all groups $\tilde{S}$ with a given alpha-cut value $\tilde{\alpha}$ from the vehicle set H. Initially lines 2 - 5 assign vehicle pairs with incentives $\geq \tilde{\alpha}$ to the groups [46, 33]. In the next step, the algorithm creates new larger groups based on initial pair-groups. In detail, lines 6 - 9 combine vehicles and existing groups to create new larger groups, taking into consideration the alpha value $\tilde{\alpha}$. The algorithm terminates if no further groups can be created for the given alpha value $\tilde{\alpha}$. Algorithm 4 groups the vehicles from the vehicle set H into $\tilde{k} \in \mathbb{N}$ groups. First, groups with a high alpha-cut value $\tilde{\alpha} = 1$ are formed by using Algorithm 3. In each iteration the algorithm checks number of formed groups. If $\tilde{k} \in \mathbb{N}$ is smaller than or equal to number of formed groups, the algorithm terminates, otherwise the algorithm reduces stepwise the alpha-cut value $\tilde{\alpha}$.

The selection of the parameters $\tilde{\alpha}$ or $\tilde{k}$ for Algorithm 3 and Algorithm 4 is not trivial. Factors such as the distribution of vehicles and the structure of the road network play a major role in the group formation. The properties such as number of groups, size of groups, disjoiness of groups and quality of groups can vary significantly. These factors have a direct influence on the computation of platooning routes (Chapter 6), both in terms of performance and quality. For low selected alpha value, Algorithm 3 will produce a large number of groups. The resulting set $\tilde{S}$ is comparable with a powerset; this contains almost all subsets of H. A computation of platooning routes for all groups (number of all groups converges to $2^{|H|} - |H| - 1$) is computationally intensive.

On the other hand, for high alpha values Algorithm 3 will form few small groups with very high platooning incentives. Many "good" groups are excluded from further computations. The computation performance of platooning routes depends on the number and size of groups.

5.2.2 Graph-Theoretical Vehicle Grouping Methods

This section looks at the vehicle grouping problem from the graph-theoretical point of view. Numerous graph-algorithms can be used to assign nodes into homogeneous groups. Here, the challenge is to choose the best strategy to find groups with the highest platooning potential. First, the incentive graph is defined as $\tilde{G} = (H, \tilde{E}, \lambda)$,

Algorithm 3 Alpha-Cut Grouping

Data: Vehicle set H and quality parameter $\tilde{\alpha}$
Result: $\tilde{S}$

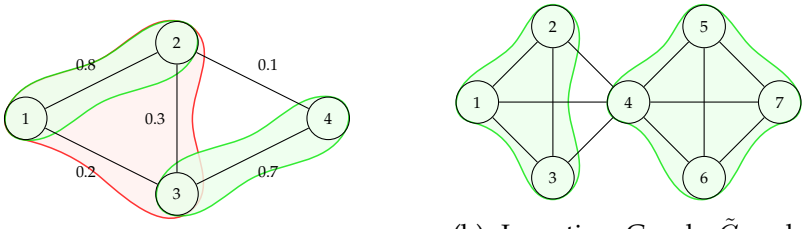
$\tilde{S} \leftarrow \emptyset$
for $i = 1 \dots |H| - 1$ **do**
  for $j = i + 1 \dots |H|$ **do**
    if $\lambda(h_i, h_j) \geq \tilde{\alpha}$ **then**
      $\tilde{S} \leftarrow \tilde{S} \cup \{h_i, h_j\}$
for each vehicle $h \in H$ **do**
  for each group $\tilde{C} \in \tilde{S}$ **do**
    if $\forall p \in \tilde{C} : \lambda(h, p) \geq \tilde{\alpha}$ **then**
      $\tilde{S} \leftarrow \tilde{S} \cup \{\tilde{C} \cup \{h\}\}$

Algorithm 4 $\tilde{k}$-Grouping

Data: Vehicle set H and number of groups $\tilde{k}$
Result: $\tilde{S}$

$\tilde{\alpha} \leftarrow 1$
$\tilde{S} \leftarrow \emptyset$
while $|\tilde{S}| < \tilde{k} \wedge \tilde{\alpha} > 0$ **do**
  $\tilde{S} \leftarrow$ Alpha-Cut Grouping$(H, \tilde{\alpha})$
  $\tilde{\alpha} \leftarrow \tilde{\alpha} - 0.1$

where H represents a set of nodes. The incentive function is an edge-weight function, which determines incentives between all vehicle pairs. The edge set $\tilde{E}$ includes only edges with function value greater than $\tilde{\alpha}$. Fig. 5.9a shows an incentive graph with four vehicles. Red colored subgraphs $\{1, 2, 3\}$ and green colored $\{\{1, 2\}$,$\{3, 4\}\}$ represent two different solutions of the vehicle grouping. The function $\tilde{w}$ (Eq. 5.17) determines the quality of a given group, e.g. $\tilde{w}(\{1, 2, 3\}) = 1.3$; the summed function values of all groups yield the solution quality. The second solution is the optimal solution with $\tilde{w}(\{1, 2\}) + \tilde{w}(\{3, 4\}) = 1.5$. A vehicle group (Definition 5.2.1) can be understood as a graph clique with at least two nodes. In general, a clique is a subset of set H, where all nodes are pairwise adjacent. Numerous clique finding algorithms can be used to determine "good" vehicle groups. On the other hand, many algorithms are not applicable for the vehicle grouping problem. Finding all maximal cliques in an undirected, unweighted graph is an important problem in graph theory and has many applications in different areas. The Bron-Kerbosch recursive algorithm [10] is widely used to solve this problem. In the case of the graph in Fig. 5.9b, the maximal cliques are $\{1, 2, 3, 4\}$ and $\{4, 5, 6, 7\}$. The finding of maximal cliques is not an optimal heuristic to group vehicles. In most cases it will produce degenerated groups with many overlaps. In the case of non-disjoint groups $\{\{1, 2, 3, 4\}, \{4, 5, 6, 7\}\}$, the algorithm in Section 6.4 will determine a disjoint group set based on group savings; vehicles $\{1, 2, 3\}$ **or** $\{4, 5, 6\}$ will not travel in a platoon. Determination of platooning routes based on a disjoint group set with maximized incentives overall, for instance, $\{\{1, 2, 3\}, \{4, 5, 6, 7\}\}$ **or** $\{\{1, 2, 3, 4\}, \{5, 6, 7\}\}$, will potentially provide better results. In other words, all vehicles can potentially be assigned to the platoons and thus more savings can be achieved. The problem can be abstracted to the Clique Partitioning Problem, which is well studied in the literature [75, 9, 36, 97, 12, 18]. The Clique Partitioning Problem describes the partitioning of node set H of graph $\tilde{G} = (H, \tilde{E}, \lambda)$ into disjoint subsets, where each subset is a



(a) Incentive Graph $\tilde{G}$ with two different grouping solutions: $\{1, 2, 3\}$ and $\{\{1, 2\},\{3, 4\}\}$.

(b) Incentive Graph $\tilde{G}$, where all edges have an incentive of 0.9. Single solution with two disjoint groups $\{\{1, 2, 3\},\{4, 5, 6, 7\}\}$.

FIGURE 5.9: Descriptive representation of Vehicle Grouping Problem in terms of Platooning.

complete graph or a clique. The goal is to find partitions with maximum weights to conform to the functions $\tilde{w}$ or $\tilde{w}'$.

The ILP (Eq. 5.20) formally describes the Clique Partitioning Problem (CPP) with maximized overall weights [75, 18]. The objective function sums up all weights or incentives selected by the binary variable $\tilde{x}_{h,p}$. If the vehicles h and p are in the same group, the variable $\tilde{x}_{h,p}$ has a value of 1. The constraint (Eq. 5.21) guarantees the disjointness of all determined groups. The ILP finds optimal groups in terms of group quality defined in Eq. 5.17.

$$\max \quad \frac{1}{2} \cdot \sum_{\substack{h,p \in H: \\ h \neq p}} \lambda(h,p) \cdot \tilde{x}_{h,p} \tag{5.20}$$

$$\text{s. t.} \quad \tilde{x}_{h,p} + \tilde{x}_{h,r} - \tilde{x}_{p,r} \leq 1 \qquad \forall \text{ distinct } h,p,r \in H \tag{5.21}$$

$$\tilde{x}_{h,p} \in \{0,1\} \qquad \forall \; h,p \in H \tag{5.22}$$

The Clique Partitioning Problem is NP-Hard. Eq. 5.20 - 5.22 describes the CPP and this formulation is used to determine the optimal solution (Optimal CPP). The solving performance of the ILP does not scale well and heuristic approaches are necessary for large problem instances.

Previously presented algorithms in Section 5.2.1 require parameters $\tilde{\alpha}$ or $\tilde{k}$. The selection of parameters is not trivial, as it depends on the road network instance and distribution of vehicles. Unsuitable parameters cause pure performance or pure results. The following approaches in *Greedy CPP* and *Exact CPP* do not necessarily depend on the parameters.

Greedy Approach based on Bron Kerbosch for Clique Partitioning Problem (*Greedy CPP*)

This section formulates greedy algorithm to solve the Clique Partitioning Problem, which uses the already introduced Bron-Kerbosch algorithm [10]. Algorithm 5 shows a detailed pseudocode of the greedy approach. For the given graph instance $\tilde{G}$ the Bron-Kerbosch algorithm determines all maximal cliques. Afterwards, the weights of the cliques are determined by the weight function $\tilde{w}$, and the clique with maximum weight will be removed from the graph $\tilde{G}$ and added to the result set $\tilde{S}$. The algorithm repeats this procedure until there are no edges remaining in

the graph. The greedy approach determines, for the graph in (Fig. 5.9a), the approximation to the optimal solution only. The *Exact CPP* uses the greedy approach to generate the initial solution.

Algorithm 5 Greedy CPP

Data: Incentive Graph $\tilde{G}$
Result: $\tilde{S}$

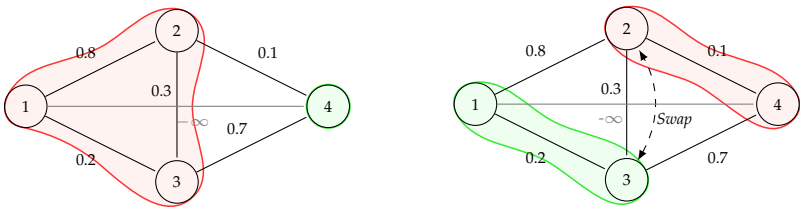
1: **while** $|\tilde{E}| > 0$ **do**
2: $\quad\tilde{C}_b \leftarrow \emptyset$
3: $\quad$**for** $\tilde{C}_c \in BronKerbosch(\tilde{G})$ **do**
4: $\quad\quad$**if** $\tilde{w}(\tilde{C}_c) > \tilde{w}(\tilde{C}_b)$ **then**
5: $\quad\quad\quad\tilde{C}_b \leftarrow \tilde{C}_c$
6: $\quad\tilde{G} \leftarrow \tilde{G}/\tilde{C}_b$
7: $\quad\tilde{S} \leftarrow \tilde{S} \cup \tilde{C}_b$

Skewed General Variable Neighborhood Search for Clique Partitioning Problem (*SGVNS CPP*)

Variable Neighborhood Search (VNS) [65, 40] is a metaheuristic which explores systematically the neighborhood structures. The approach jumps randomly from the neighborhood of the current solution to another one iff the local search method finds any improvements in the new neighborhood. Generally, the approach consists of three steps *shaking*, *local search*, and *neighborhood change*. The *shaking* step randomly produces a new solution from the neighborhood of the current solution. The *local search* finds a local optimum based on a previously generated solution. In the last step, *neighborhood change*, the current solution is compared with the new local optimal solution. If the new solution is better than the current solution, it will be set as the current solution and the neighborhoods of this new current solution will be explored. Otherwise, the algorithm further examines the neighborhoods of current solutions. If no improvements are obtained after a predefined number of examinations, the algorithm terminates. The General Variable Neighborhood Search (GVNS) (VNS/VND) method is an extended version of the VNS metaheuristic, which replaces the *local search* step with the Variable Neighborhood Descent (VND) method. The VND method is very similar to the VNS approach, except that it does not contain the *shaking* step and works deterministically [83, 66]. The Skewed General Variable Neighborhood Search (SGVNS) algorithm is an extension of the GVNS method. The worse solution produced by GVNS iteration can be accepted, if the solution is sufficiently different from the current solution [8, 62, 9].

Authors in [9] used the SGVNS approach to solve the CPP; Algorithm 6 describes the SGVNS framework. The algorithm requires an initial solution $\tilde{x}_i$, e.g. a trivial solution, in which each element belongs to any clique. First, the VND algorithm is applied to determine the local maximum solution based on the initial solution and set determined solution as the current solution $\tilde{x}_c$ and the best solution $\tilde{x}_b$ (lines 1 - 2). The VND method explores the *Insertion* and *Swap* neighborhood structures of the initial solution $\tilde{x}_i$ until the local maximum is found, according to the weight function $\tilde{w}$. The *Insertion* method removes one element from the current clique and inserts these into the other clique; all remaining cliques and their weights stay unchanged. This procedure is repeated for all elements and tries to insert them into existing cliques or create a new clique with the given element. Fig. 5.10a shows an example in which the node 3 moves, from the red clique to the green clique. The

weights of both cliques are $0.8 + 0.2 + 0.3 + 0 = 1.3$. After the insertion, the new cliques are formed $\{\{1,2\}, \{3,4\}\}$, which represent a local maximum solution with the following weights $0.8 + 0.7 = 1.5$. The *Swap* method swaps two elements from two different cliques; all remaining cliques stay unchanged. Fig. 5.10b illustrates a swap operation, in which nodes 2 and 3 from two different cliques are swapped. The sum of the weights is improved from 0.3 to 1.5. The VND executes the *Insertion* method until no improvements are found; then the algorithm runs the *Swap* method once. If the swap improves the solution, the *Insertion* method will be executed again until no improvements can be obtained; otherwise, the VND terminates. The fundamental part of the SGVNS framework is a loop (line 3), which explores new *Insertion* and *Swap* neighborhood structures without trajectory until the stopping condition is met; i.e. the maximum number of iterations is reached [65]. Based on the current solution $\tilde{x}_c$, the *shaking* function produces a new solution $\tilde{x}_n$, which randomly selects two nodes from different cliques and swaps them. The number of "shakes" can be selected flexibly. The VND in the line 5 uses a new solution to find the local optimum and to renew the solution $\tilde{x}_n$. Line 6 compares the new solution $\tilde{x}_n$ with the current $\tilde{x}_c$ and the best solution $\tilde{x}_b$. If the new solution $\tilde{x}_n$ is better or sufficiently different from the $\tilde{x}_c$ and $\tilde{x}_b$ solutions, then the new solution $\tilde{x}_n$ is set as the current solution $\tilde{x}_c$ (line 7). The quality of the obtained solution can be determined by the objective function $\tilde{f}$, corresponding to the weighting function $\tilde{w}$. The variance between two solutions is measured by the distance function $\tilde{d}$. The function counts edges which are available in the first solution and not in the second solution, and vice versa, edges which are not available in the first solution but in the second one. Counted edges are divided by the number of edges which are included in the first solution. The importance of the difference between the solutions can be additionally controlled by parameter $\tilde{\beta}$. If the new solution $\tilde{x}_n$ is in fact better than the actual best solution $\tilde{x}_b$, then the algorithm sets the new solution as the best solution $\tilde{x}_b \leftarrow \tilde{x}_n$. After the termination, the algorithm returns the best solution $\tilde{x}_b$ with corresponding cliques. Cliques with more than one element are added to set $\tilde{S}$.



(a) *Insertion* operation, which inserts element 3 from clique $\{1,2,3\}$ to clique $\{4\}$.

(b) *Swap* operation, which swaps elements 2 and 3 from the following cliques $\{\{1,2\},\{3,4\}\}$.

FIGURE 5.10: Neighborhoods structures used by VND approach.

Different methods to calculate initial solutions are proposed in the literature [8, 83, 41]; i.e. the random assignment of elements to the groups or different greedy approaches. Depending on the initial solution, the SGVNS algorithm produces solutions with various qualities. In this work, three different methods are used to calculate initial solutions. The trivial method assigns each element exactly to one group. The random method assigns the elements randomly to groups. The greedy method is already described in Section 5.2.2.

Algorithm 6 Skewed General Variable Neighborhood Search for CPP [9]

Data: Incentive graph $\tilde{G}$ and initial solution $\tilde{x}_i$
Result: $\tilde{S}$

$\tilde{x}_c \leftarrow \text{VND}(\tilde{x}_i)$
$\tilde{x}_b \leftarrow \tilde{x}_c$
while *stopping condition* **do**
    $\tilde{x}_n \leftarrow shaking(\tilde{x}_c)$
    $\tilde{x}_n \leftarrow \text{VND}(\tilde{x}_n)$
    if $\tilde{f}(\tilde{x}_n) + \tilde{\beta}\tilde{d}(\tilde{x}_n, \tilde{x}_c)|\tilde{f}(\tilde{x}_c)| > \tilde{f}(\tilde{x}_c)$ **and** $\tilde{f}(\tilde{x}_n) + \tilde{\beta}\tilde{d}(\tilde{x}_n, \tilde{x}_b)|\tilde{f}(\tilde{x}_b)| > \tilde{f}(\tilde{x}_b)$ **then**
        $\tilde{x}_c \leftarrow \tilde{x}_n$
        if $\tilde{f}(\tilde{x}_n) > \tilde{f}(\tilde{x}_b)$ **then**
            $\tilde{x}_b \leftarrow \tilde{x}_n$
$\tilde{S} \leftarrow convert(\tilde{x}_b)$

5.2.3 Disjointness Relaxation of Vehicle Group Set

The previously discussed approaches Eq. 5.20, Algorithm 5 and Algorithm 6 provide disjoint vehicle groups with overall maximized incentives. The incentives of the groups give an estimate of how "good" the grouped vehicles can form profitable platoons. Fig. 5.9b represents graph $\tilde{G}$ in which two different optimal solutions can be obtained, $\tilde{w}(\{1,2,3\}) + \tilde{w}(\{4,5,6,7\}) = 8.1$ and $\tilde{w}(\{1,2,3,4\}) + \tilde{w}(\{5,6,7\}) = 8.1$. Savings between the shown solutions may vary depending on the actual structure of the road network instance. The intended extension of a disjoint group set, e.g. $\{\{1,2,3\},\{4,5,6,7\},\{1,2,3,4\}\}$ will initiate the additional route computations (Section 6) for added groups. On the other hand, the relaxation of disjointness can improve the formation of vehicle platoons and provides higher fuel savings. The trade-off between computational complexity and quality of the results must be found. This section introduces a measurement and parameterization of disjointness for a given group set. Similar ideas were used in determining alternative routes [1, 61], in which the alternative path shares only a limited number of edges of the shortest path. The disjointness of paths can be controlled by a parameter. Authors in [2, 5] relax the disjointness constraint for the Disjoint Cycle Packing Problem.

$$\phi : \tilde{S} \mapsto \frac{|\bigcup \tilde{S}|}{\sum\limits_{\tilde{C} \in \tilde{S}} |\tilde{C}|} \quad \tilde{S} \neq \emptyset \tag{5.23}$$

The disjointness of a group set $\tilde{S}$ can be measured by the function ϕ (Eq. 5.23). $|\bigcup \tilde{S}|$ counts individual vehicles in set $\tilde{S}$ divided by the sum of vehicles in certain groups. The following example $\tilde{S} = \{\{1,2,3\},\{4,5,6,7\}\}$ represents a disjoint group set. In this case the function returns the following value $\phi(\tilde{S}) = \frac{7}{7} = 1$. The function value for the non-disjoint group set is < 1, e.g. $\phi(\{\{1,2,3\},\{4,5,6,7\},\{1,2,3,4\}\}) = \frac{7}{11} \approx 0.63$.

$$\tilde{K} = (\tilde{S}_1, \ldots, \tilde{S}_k) \quad k \in \mathbb{N} \tag{5.24}$$

Eq. 5.24 defines a tuple $\tilde{K}$ with k-best group sets $\tilde{S}$, in which each particular set $\tilde{S}$ contains disjoint groups. Sets in tuple $\tilde{K}$ are sorted in descending order; groups in set $\tilde{S}_1$ have the most incentives in total. Algorithm 6 can be easily adjusted to obtain k-best solutions.

Algorithm 7 determines group set $\tilde{S}$ based on the tuple $\tilde{K}$ with parameterized disjointness $\phi(\tilde{S}) \geq \tilde{\sigma}$. Initially, the algorithms assign the best solution $\tilde{S}_1$ from tuple $\tilde{K}$ to set $\tilde{S}$. All other groups from the tuple $\tilde{K}$ will be united, sorted in descending order by group incentives and added to the tuple $\tilde{S}_{sort}$ (line 2). The algorithm adds single groups, which satisfy the condition in line 4 to set $\tilde{S}$.

Algorithm 7 Group Set Disjointness Relaxation

Data: $\tilde{K}, \tilde{\sigma}$
Result: $\tilde{S}$
1: $\tilde{S} \leftarrow \tilde{S}_1$
2: $\tilde{S}_{sort} \leftarrow Sort(\bigcup \tilde{K} \setminus \tilde{S}_1)$
3: **for** $\tilde{C} \in \tilde{S}_{sort}$ **do**
4: **if** $\phi(\tilde{S} \cup \tilde{C}) \geq \tilde{\sigma}$ **then**
5: $\tilde{S} \leftarrow \tilde{S} \cup \tilde{C}$

5.3 Resolution of Non-Disjoint Groups

5.3.1 Vehicle Assignment Partitioning

The previously presented approaches contain certain weaknesses caused by disjoint group formation (Section 5.2). Fig. 5.11 illustrates the disjointness problem; vehicle h can be assigned into two non-disjoint groups $\{h, h_1\}$ and $\{h, h_2\}$. The vehicles h_1 and h_2 are not combinable; consequently, the group and the platoon with vehicles $\{h, h_1, h_2\}$ cannot be formed. A potential solution is to cut the shortest path of vehicle h into multiple partitions (Def. 5.3.1), e.g. $h^{(1)}$ and $h^{(2)}$ to form the groups $\{h^{(1)}, h_1\}$ and $\{h^{(2)}, h_2\}$, and on this basis, the near-optimal platoons. Finding such partitions is defined in this work as *Route Partitioning Problem*.

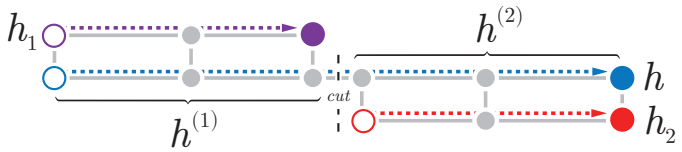


FIGURE 5.11: Route partitioning of vehicle h.

Definition 5.3.1 (Route Partitioning Solution)**.** The Route Partitioning Solution is a tuple of partitions of one vehicle assignment $h \in H$. Partitions are vehicle assignments and concatenating their shortest paths produces a path for vehicle h. In this work, the concatenation yields the shortest path of h. Eq. 5.25 shows κ partitions of the vehicle assignment h.

$$(h^{(1)}, h^{(2)}, h^{(3)}, \ldots, h^{(\kappa)}) \tag{5.25}$$

Identifying and solving such problems in the grouping process is a significant challenge; incorrectly partitioned vehicles can degrade the computation performance and the fuel consumption. This problem is divided into two subproblems: identification of suitable vehicles for partitioning and finding of meaningful route cuts for selected vehicles.

Identification of Suitable Vehicles for Partitioning

A vehicle suitable for partitioning is necessarily assigned in several groups, as shown for vehicle h in Fig. 5.11. A further requirement for partitioning is the length of the shortest path P_h; only vehicle assignments with large distances should be partitioned. The vehicle set Q in Eq. 5.26 includes such vehicles. The minimum distance for platooning in partitioning context is specified by the parameter ξ_1. Vehicles with small distances can platoon, if

$$Q = \{h \in H : |\{\tilde{C} \in \tilde{S} : h \in \tilde{C}\}| > 1 \wedge c(P_h) > 2 \cdot \xi_1\} \tag{5.26}$$

The vehicle set Q also only consists of vehicles that have a high incentive to at least one other vehicle. The incentive lower bound is specified by parameter ξ_2.

$$Q = \{h \in Q : \exists p \in H : \lambda(h,p) > \xi_2 \wedge h \neq p\} \tag{5.27}$$

Partitioning of Vehicle Assignment

Partitioning of selected vehicles Q is a very complex task, because it requires a closer look at the surrounding vehicles. For each vehicle h from Q, a set Λ_h is determined (Eq. 5.28), which holds relevant vehicles for finding cutting positions, see Fig. 5.11. The relevance is specified by incentive ξ_3 and distance ξ_1 parameters.

$$\Lambda_h = \{p \in H : \lambda(h,p) > \xi_3 \wedge c(P_p) > \xi_1 \wedge h \neq p\} \quad \forall h \in Q \tag{5.28}$$

Eq. 5.29 introduces a new measurement function λ'_h, which ranks the vehicles Λ_h, in regard to all vehicles $h \in Q$. Def. 5.3.2 formulates a new geometric container for vehicles in the set Q. The function returns the normalized value of the intersection area of VGCC $\Gamma(P_p)$ and VGC $\widetilde{\Gamma}(P_h)$, which is used to determine relevant vehicles for partitioning.

$$\lambda'_h(p) = \frac{A(\widetilde{\Gamma}(P_h) \cap \Gamma(P_p))}{A(\widetilde{\Gamma}(P_h))} \quad h \in Q, p \in \Lambda_h \tag{5.29}$$

Definition 5.3.2 (Vehicle Geometric Container (VGC))**.** The Vehicle Geometric Container is a subset in $\mathbb{R}^2$, which is not necessarily a convex subset. The area consists of points that are located near of the shortest path (Eq. 5.31). The function $\dot{P}(P)$ provides the geometric points of the shortest path (Eq. 5.30), which are used to construct the $\widetilde{\Gamma}(P)$ VGC. $\widetilde{\Gamma}(P)$ is illustrated in Fig. 5.12 (blue dashed line).

$$\dot{P}(P) = \bigcup_{(v,u) \in P} \{\Theta \cdot \gamma(v) + (1 - \Theta) \cdot \gamma(u) : \Theta \in [0,1]\} \tag{5.30}$$

$$\widetilde{\Gamma}(P) = \{x \in \mathbb{R}^2 : \exists y \in \dot{P}(P) : d(x,y) \leq \eta'' \cdot c(P)\} \tag{5.31}$$

The main challenge of this heuristic is to divide the shortest path of the considered vehicle h from set Q and to form more profitable groups afterwards. For this purpose, the ILP (Eq. 5.32) uses the predefined function λ'_h to select "conflict-free" and suitable vehicles from the set Λ_h. Subsequently, the selected vehicles are used to find the cutting points and cutting nodes on the shortest path. The objective function selects relevant vehicles according to the function λ'_h. In Fig. 5.12 $h \in Q$ vehicle is assigned to two groups: $\{h, p_1, p_2\}$ and $\{h, p_3, p_4, p_5\}$. The constraint (Eq. 5.33) ensures that the selected vehicles are conflict-free. The vector projections of vehicle vectors $\underline{p}_1, \ldots, \underline{p}_5$ onto vehicle h are represented by vectors (black). Iff the vector projections

do not overlap, then the corresponding vehicles are conflict-free, e.g. following vehicle combinations do not conflict $\{p_1, p_4\}, \{p_2, p_3\}$.

$$\max \quad \sum_{p \in \Lambda_h} \lambda'_h(p) \cdot x_p \tag{5.32}$$

$$\text{s. t.} \quad x_{p_1} + x_{p_2} \leq 1 \qquad \forall \;\; p_1, p_2 \in \Lambda_h : a_{p_1} <_h b_{p_2} \wedge b_{p_2} <_h b_{p_1} \tag{5.33}$$

$$x_p \in \{0, 1\} \qquad \forall \;\; p \in \Lambda_h \tag{5.34}$$

Formulas in Eq. 5.35 - 5.37 introduce the relation $v <_h u$ between two nodes with regard to the vehicle h. Fig. 5.13 shows an example, in which the node u is greater than the node v. The first step is to determine an intersection point of the line with a start point a and direction vector $\underline{h}$ and a line with start point v and direction vector $\underline{h}_\perp$. The intersection point of u is determined in the same way.

The distance from the start point a to the corresponding intersection points is compared by the relation $<$. Eq. 5.35 defines the logical equivalence between $<$ and $<_h$.

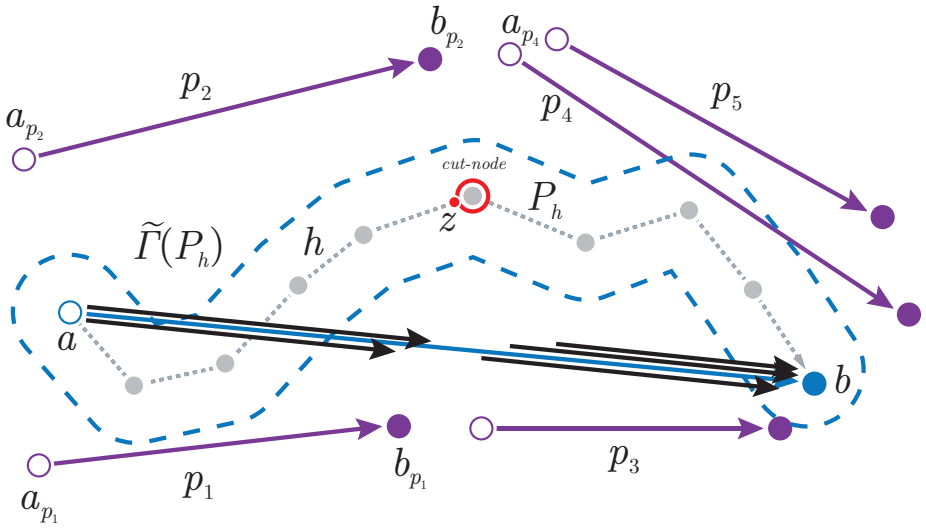


FIGURE 5.12: Platooning with surgery: Example of a conflict group and resolution of the conflict by the partitioning approach.

The Constraint (Eq. 5.35) considers all conflicted vehicle pairs in the set Λ_h and allows one vehicle per pair only. Vehicles p_1 and p_2 in Fig. 5.12 overlap, as end node b_{p_1} of vehicle p_1 is between start node a_{p_2} and end node b_{p_2} of vehicle p_2. In the case of the overlapping vehicles p_1 and p_2, vehicle p_1 is selected, because the function value of vehicle p_1 is greater than the value of vehicle p_2 ($\lambda'_h(p_1) > \lambda'_h(p_2)$).

$$a_h + x^*_v \cdot \underline{h} = \gamma(v) + y^*_v \cdot \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \underline{h} \tag{5.35}$$

$$a_h + x^*_u \cdot \underline{h} = \gamma(u) + y^*_u \cdot \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \cdot \underline{h} \tag{5.36}$$

$$x^*_v < x^*_u \Leftrightarrow v <_h u \qquad\qquad v, u \in V \tag{5.37}$$

The optimal solution obtained by ILP (Eq. 5.32) for the given example consists of vehicles p_1 and p_4. The cut-node can be derived from each conflict-free vehicle pair, by end and start node. The intersection of the line from b_{p_1} to a_{p_4} and point set $\dot{P}(P_h)$ yields a geometric point z. The closest node on the shortest path P_h to z is a *cut-node*.

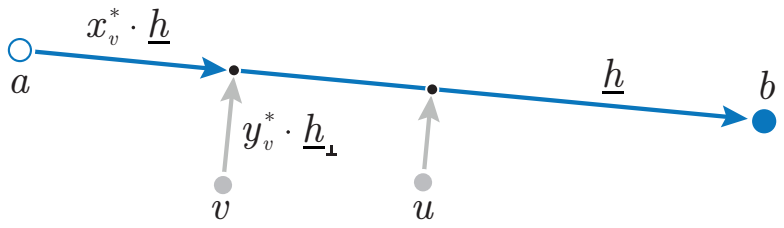


FIGURE 5.13: Example of the relation $<_h$ defined in Eq. 5.35 - 5.37.

5.4 Summary

This chapter introduces the grouping-based routing algorithms. The Section 5.1 describes the incentives methods, which calculates pairwise the platooning opportunities. Section 5.2 provides the α-*cut Method*, *Greedy CPP*, *Exact CPP* and *SGVNS CPP* methods to form the candidate groups based on pairwise incentives. Section 5.3 outlines approaches to resolve the non-disjoint groups.

Chapter 6

Computation of near-optimal Platoon Routing based on the Candidate Groups

This chapter presents different approaches to find optimal or near-optimal platoon routing. Section 6.1 discusses the ILP formulation of the Vehicle Platooning Problem briefly. Section 6.2 proposes our novel Column Generation formulation for the Vehicle Platooning Problem. Section 6.3 revisits the state-of-the-art algorithm Hub Heuristic [57] and gives a new formulation for it. Furthermore, the time scheduling algorithm is introduced in Section 6.5 to determine the travel times of vehicles in platoons.

6.1 An Exact Solver for the Vehicle Platooning Problem

The previous chapter introduced methods to group vehicles and reduced the search space in the road networks by using geometric containers. Each group represents a valid problem instance of the VPP. This section introduces an *Exact Solver* for the VPP based on the ILP formulation in Eq. 2.8 - Eq. 2.12. A similar formulation was originally proposed in [57]. The *Exact Solver* solves the VPP with the well known *Simplex* and *Branch-and-Bound* methods [37]. The integrality restriction of the ILP formulation must be removed to use the simplex method. After solving the relaxed ILP, the *Branch-and-Bound* is used to determine an integer solution by branching the variables with a continuous value. The Gurobi Optimizer [64] is used to solve the ILP formulation. For larger problem instances, the *Exact Solver* shows poor scalability [88]. Fig. 6.1 illustrates the convergence behavior of the solving process for the problem instance with 200 vehicles on the 30×30 grid road network. The orange graph shows the objective values of performed dual simplex iterations. The rate of convergence of the *Dual Simplex* method slows down after 5700 seconds, and the optimal solution is found after 14263 seconds. Usually, only a few *Branch-and-Bound* iterations are required to determine the optimal integer solution. For large problem instances, the *Exact Solver* runs out of memory, see Table 8.9.

6.2 Platooning Routing with Column Generation (CG)

Column generation is a technique for solving specific linear programs where the number of variables is much larger than the number of constraints [22, 19, 17]. Most of the variables are non-basic and take a value of 0 in the optimal solution. The general idea is to restrict the optimization problem by considering only a **small subset**

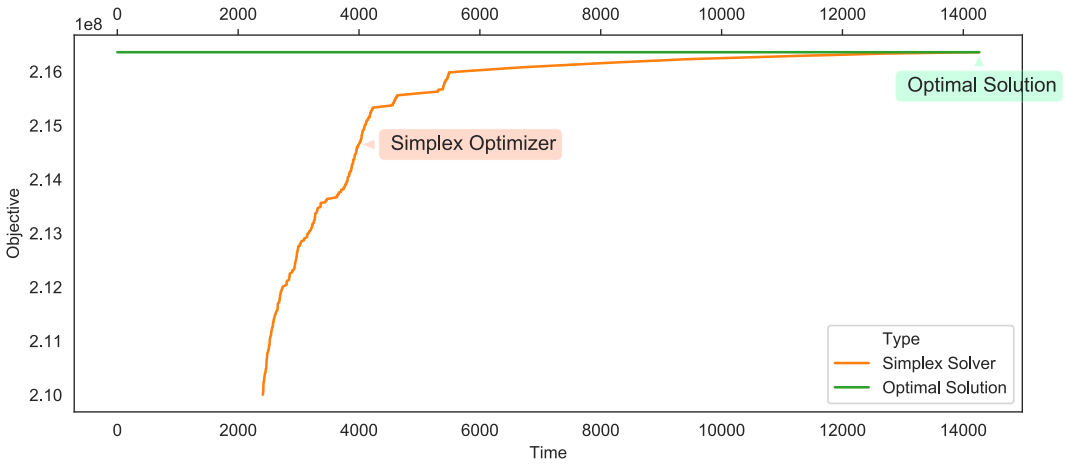


FIGURE 6.1: Objective values of dual simplex iterations of the ILP in Eq. 2.8 - Eq. 2.12 performed by Gurobi Optimizer [64].

of variables. Iteratively, **new variables (columns)** are added to the restricted optimization problem, which potentially improves the current solution. The Column Generation formulation of VPP was initially proposed in [14]; this section further elaborates on the formulation.

The ILP (Eq. 2.8 - Eq. 2.12) is not suitable for the column generation algorithm since the program has relatively few variables ($|H| + |E| \cdot |H|$) and many constraints ($|V| \cdot |H| + |E| \cdot |H|$). The **equivalently transformed** ILP, called **Master Problem**, is given in Eq. 6.1 - Eq. 6.5. The important difference is that the Master Problem defines a variable x_P for each path instead of a variable x_{eh} for each edge and vehicle. The set GP_h contains all feasible paths of a vehicle h. Constraint Eq. 6.2 enforces that exactly one path is assigned to exactly one vehicle. Constraint Eq. 6.3 enables the variable y_e if the corresponding edge e is used in any selected path. The cardinality of the path set GP_h can be very large, even for a small road network. Our column generation algorithm starts with a small and manageable path set GP_h for each vehicle, solves the subproblem, analyzes the solution, generates and adds new vehicle paths to the path set GP_h, and repeats the iteration untill the solution cannot be enhanced.

The flowchart (Fig. 6.2) depicts the column generation algorithm for solving VPP. In the beginning, different algorithms can be used to produce an initial feasible solution, which matches the following constraints Eq. 6.2 - Eq. 6.5. A straightforward approach is, for example, to calculate the shortest path for each vehicle, so that $GP_h = \{P_h\} \quad \forall h \in H$. The **Restricted Master Problem (RMP)** considers only the initially generated shortest paths so that the number of variables is $|H| + |E|$.

$$\min \quad \sum_{e \in E} \eta \cdot c(e) \cdot y_e + \sum_{h \in H} \sum_{P \in GP_h} (1 - \eta) \cdot c(P) \cdot x_P \tag{6.1}$$

$$\text{s. t.} \quad \sum_{P \in GP_h} x_P = 1 \qquad \forall h \in H \tag{6.2}$$

$$\sum_{\substack{P \in GP_h: \\ e \in P}} x_P \leq y_e \qquad \forall e \in E, h \in H \tag{6.3}$$

$$x_P \in \{0, 1\} \qquad \forall P \in GP_h \quad \forall h \in H \tag{6.4}$$

$$y_e \in \{0, 1\} \qquad \forall e \in E \tag{6.5}$$

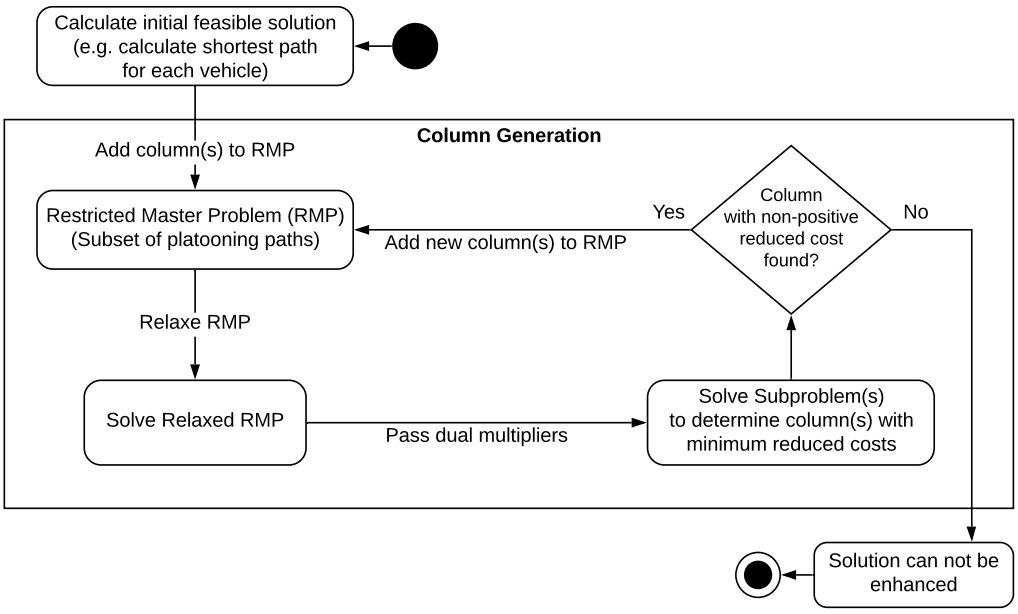


FIGURE 6.2: Column generation algorithm for solving the VPP

A detailed consideration of the RMP is necessary to formulate the Subproblem (SP). Eq. 6.6 - Eq. 6.10 shows the RMP in standard form. Eq. 6.2 is simplified and only expresses that at least one path must be assigned to a vehicle since the savings η are less than or equal to 1. The corresponding constraints Eq. 6.4 and Eq. 6.5 are relaxed to $x_P \geq 0$ and $y_e \geq 0$ so that the problem is **relaxed**.

$$\max \quad -\sum_{e \in E} \eta \cdot c(e) \cdot y_e - \sum_{h \in H} \sum_{P \in GP_h} (1 - \eta) \cdot c(P) \cdot x_P \tag{6.6}$$

$$\text{s. t.} \quad -\sum_{P \in GP_h} x_P \leq -1 \qquad \forall h \in H \tag{6.7}$$

$$\sum_{\substack{P \in GP_h: \\ e \in P}} x_P - y_e \leq 0 \qquad \forall e \in E, h \in H \tag{6.8}$$

$$x_P \geq 0 \qquad \forall P \in GP_h \quad \forall h \in H \tag{6.9}$$

$$y_e \geq 0 \qquad \forall e \in E \tag{6.10}$$

Eq. 6.11 shows the general vector formulation of the relaxed RMP, vector $\underline{x}$ comprises all decision variables, vector $\underline{c}$ comprises the cost coefficients, and vector $\underline{b}$ comprises the righthand-side coefficients. This simplification helps to keep further explanations compact.

$$\underline{x} = \begin{pmatrix} y_1 \\ \vdots \\ y_{|E|} \\ x_1 \\ \vdots \\ x_{\sum_{h \in H} |GP_h|} \end{pmatrix}, \quad \underline{c} = \begin{pmatrix} -\eta \cdot c(e_1) \\ \vdots \\ -\eta \cdot c(e_{|E|}) \\ (\eta - 1) \cdot c(P_1) \\ \vdots \\ (\eta - 1) \cdot c(P_{\sum_{h \in H} |GP_h|}) \end{pmatrix}, \quad \underline{b} = \begin{pmatrix} \hat{b}_1 = -1 \\ \vdots \\ \hat{b}_{|H|} = -1 \\ \hat{b}_{|H|+1} = 0 \\ \vdots \\ \hat{b}_{|H|+|E| \cdot |H|} = 0 \end{pmatrix} \tag{6.11}$$

Eq. 6.12 depicts one column of the coefficient matrix $\hat{A}$, which represents a particular path. The upper part of the column specifies to which vehicle this column belongs. The green marked coefficient $\hat{a}_2$ has a value of -1 for $h_2 \in H$ and others have value 0, it means the current path belongs to the vehicle h_2. The lower part of the column defines which vehicles use which edges, e.g., for h_2 only the green marked coefficients (i.e., edges) are relevant. Coefficients of used edges in the path have the value 1, and others have the value 0. All columns generated by subproblems have the same format.

$$\begin{pmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_{|H|} \\ \hline \hat{a}_{1,1} \\ \vdots \\ \hat{a}_{|E|,1} \\ \hat{a}_{1,2} \\ \vdots \\ \hat{a}_{|E|,2} \\ \vdots \\ \hat{a}_{1,|H|} \\ \vdots \\ \hat{a}_{|E|,|H|} \end{pmatrix} \tag{6.12}$$

Subproblem derivation: Eq. 6.13 depicts the RMP in general form with $\underline{x}, \underline{c}$, and $\underline{b}$ vectors and coefficient matrix $\hat{A}$.

$$\begin{aligned} \max \quad & \underline{c}^\intercal \underline{x} \\ \text{s. t.} \quad & \hat{A}\underline{x} \leq \underline{b} \\ & \underline{x} \geq 0 \end{aligned} \tag{6.13}$$

The problem is transformed into a normal form by adding slack variables. Furthermore, the problem is separated into the basic B and non-basic N vectors and matrices.

$$\begin{aligned} \max_{\underline{x}_B, \underline{x}_N} \quad & \underline{c}_B^\intercal \underline{x}_B + \underline{c}_N^\intercal \underline{x}_N \\ \text{s. t.} \quad & \hat{A}_B \underline{x}_B + \hat{A}_N \underline{x}_N = \underline{b} \\ & \underline{x}_B \geq 0, \underline{x}_N \geq 0 \end{aligned} \tag{6.14}$$

The idea of column generation is to analyze the current solution $\underline{x}_B$ and make a statement if the current solution can be improved. For a specific solution, ILP 6.14 can be transformed into ILP 6.16 with Eq. 6.15.

$$\underline{x}_B = \hat{A}_B^{-1}(\underline{b} - \hat{A}_N \underline{x}_N) \tag{6.15}$$

$$\begin{aligned} \max_{\underline{x}_N} \quad & \underline{c}_B^\intercal \hat{A}_B^{-1} \underline{b} + (\underline{c}_N^\intercal - \underline{c}_B^\intercal \hat{A}_B^{-1} \hat{A}_N) \underline{x}_N \\ \text{s. t.} \quad & \hat{A}_B^{-1}(\underline{b} - \hat{A}_N \underline{x}_N) \geq 0 \\ & \underline{x}_N \geq 0 \end{aligned} \tag{6.16}$$

The optimization problem in Eq. 6.16 depends only on the variables in $\underline{x}_N$. Problem constraint enforces that the current solution $\underline{x}_B$ is feasible. The orange marker is the objective function value of the considered solution $\underline{x}_B$. The term marked green determines **reduced costs** for non-basic variables $\underline{x}_N$ based on **shadow prices**. The shadow prices are calculated with basic coefficients of the objective function $\underline{c}_B$ and basic coefficient matrix $\hat{A}_B$ shown in Eq. 6.17. Each constraint in Eq. 6.2 has exactly one dual variable in $\left(\pi_1 \cdots \pi_{|H|}\right)^\intercal$ and each constraint in Eq. 6.3 has exactly one dual variable in $\left(\pi_1 \cdots \pi_{|E||H|}\right)^\intercal$.

$$\underline{c}_B^\intercal \cdot \hat{A}_B^{-1} = \pi = \begin{pmatrix} \pi_1 \\ \vdots \\ \pi_{|H|} \\ \pi_{11} \\ \vdots \\ \pi_{|E||H|} \end{pmatrix} \tag{6.17}$$

For each vehicle $h \in H$, exactly one **subproblem** is formulated in Eq. 6.18 - Eq. 6.20. The goal is to generate a new path with **negative** reduced costs to decrease the current fuel consumptio n; the reduced costs can be expressed as follows:

reduced costs $=$ *new path costs* $+$ *current savings* $-$ *current path costs* $-$ *new savings*

The subproblem is the shortest path problem, and the objective function minimizes the **reduced costs**. If the reduced costs $\bar{c}^*$ are below 0, then the corresponding generated path improves the current solution. The constraint in Eq. 6.19 enforces flow conservation.

$$\bar{c}_h^* = \min \sum_{e \in E} (c(e) - \sum_{h' \in H \setminus h} \pi_{eh'}) \cdot x_e - \pi_h \tag{6.18}$$

$$\text{s. t.} \sum_{e \in \delta^+(v)} x_e - \sum_{e \in \delta^-(v)} x_e = \begin{cases} 1 & \text{if } v = a \quad \forall v \in V \\ -1 & \text{if } v = b \quad \forall v \in V \\ 0 & \text{otherwise} \end{cases} \tag{6.19}$$

$$x_e \in \{0, 1\} \qquad \forall e \in E \tag{6.20}$$

Algorithm 8 summarizes our column generation method to solve the VPP. Lines 2 - 5 show the initial generation of feasible paths for given vehicles. Lines 6 - 13 represent the main part of the algorithm, which is repeated until no better solution can be obtained. The shadow prices are determined by solving the relaxed RMP (line 8). They are used to formulate subproblems. For each vehicle, the algorithm solves the previously defined subproblems (line 9). If a **new** path with reduced costs $\bar{c}^* \leq 0$ is found (line 11), this path is added to the set GP_h, and a further iteration is initiated. Finally, if no new column can be generated, the algorithm determines a feasible solution by solving the RMP (line 14), which includes precisely $|H|$ paths.

Algorithm 8 Column Generation to solve VPP

Data: Road network instance $G = (V, E)$ and vehicle set H
Result: (Optimal) platoon routing $O = \{P_1^{pl}, \ldots, P_{|H|}^{pl}\}$ for vehicle set H.

1: **for** $h \in H$ **do**
2: $GP_h \leftarrow \emptyset$
3: $P_h =$ *initial path calculation*(G, h)
4: $GP_h \leftarrow GP_h \cup P_h$
5: *run = True*
6: **while** *run* **do**
7: *run = False*
8: $\pi =$ *solve relaxed RMP (Eq. 6.6 - Eq. 6.10)*
9: **for** $h \in H$ **do**
10: $\bar{c}^*, P_{new} =$ *solve SP (Eq. 6.18 - Eq. 6.20)*
11: **if** $\bar{c}^* \leq 0$ & $P_{new} \notin GP$ **then**
12: $GP_h \leftarrow GP_h \cup P_{new}$
13: *run = True*
14: $O =$ *solve RMP (Eq. 6.1 - Eq. 6.5)*

A practical example in Fig. 6.3 and Fig. 6.4 illustrates the operation of the column generation method. In our example, the four vehicles travel independently of each other in the initial solution (Fig. 6.3). Three iterations are necessary to determine the optimal solution, and after the third iteration, the algorithm terminates. Table 6.1 shows detailed information for each iteration. Descriptions are divided into MP and SP. Row P_{cur} shows the current paths for each vehicle after solving the MP. Newly generated paths by the subproblems are specified in row P_{new}, the corresponding path costs minus savings are listed by $\bar{c}(P_{new})$. The shadow prices π_h represent the current path costs minus current savings of the considered vehicle h. The reduced costs $\bar{c}^*$ indicate whether the newly generated path can improve the current solution or not. In each iteration, the reduced costs are negative (green marked) except for the last iteration. Unchanged paths and prices are grayed out.

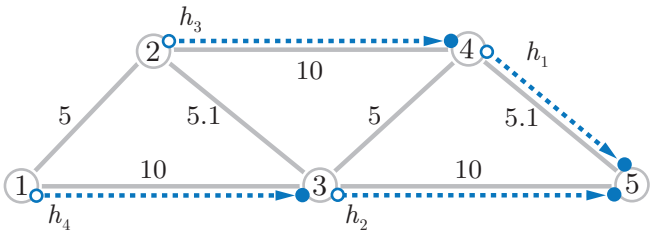


FIGURE 6.3: Example VPP instance with four vehicles h_1, h_2, h_3, and h_4 to illustrate our proposed column generation method. The objective value of the initial solution is $\bar{F}(x) = 35.1$.

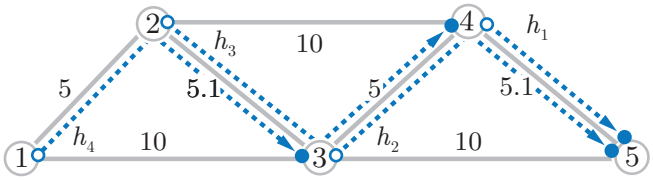


FIGURE 6.4: Optimal platooning routing solved by our proposed column generation method. The objective value of the optimal solution is $\bar{F}(x) = 33.88$ with $\eta = 0.1$.

TABLE 6.1: Details of column generation example.

			h_1	h_2	h_3	h_4	$\bar{F}(x)$
1	*MP*	P_{cur}	$\{(4,5)\}$	$\{(3,5)\}$	$\{(2,4)\}$	$\{(1,3)\}$	
	SP	P_{new}	$\{(4,5)\}$	$\{(3,4),(4,5)\}$	$\{(2,4)\}$	$\{(1,3)\}$	
		$\bar{c}(P_{new})$	5.1	9.59	10	10	34.69
		π_h	5.1	10	10	10	
		$\bar{c}^*$	0.0	-0.41	0.0	0.0	
2	*MP*	P_{cur}	$\{(4,5)\}$	$\{(3,4),(4,5)\}$	$\{(2,4)\}$	$\{(1,3)\}$	
	SP	P_{new}	$\{(4,5)\}$	$\{(3,4),(4,5)\}$	$\{(2,3),(3,4)\}$	$\{(1,3)\}$	
		$\bar{c}(P_{new})$	5.1	9.59	9.6	10	34.29
		π_h	5.1	9.59	10	10	
		$\bar{c}^*$	0.0	0.0	-0.4	0.0	
3	*MP*	P_{cur}	$\{(4,5)\}$	$\{(3,4),(4,5)\}$	$\{(2,3),(3,4)\}$	$\{(1,3)\}$	
	SP	P_{new}	$\{(4,5)\}$	$\{(3,4),(4,5)\}$	$\{(2,3),(3,4)\}$	$\{(1,2),(2,3)\}$	
		$\bar{c}(P_{new})$	5.1	9.59	9.6	9.59	33.88
		π_h	5.1	9.59	9.6	10	
		$\bar{c}^*$	0.0	0.0	0.0	-0.41	
4	*MP*	P_{cur}	$\{(4,5)\}$	$\{(3,4),(4,5)\}$	$\{(2,3),(3,4)\}$	$\{(1,2),(2,3)\}$	
	SP	P_{new}	$\{(4,5)\}$	$\{(3,4),(4,5)\}$	$\{(2,3),(3,4)\}$	$\{(1,2),(2,3)\}$	
		$\bar{c}(P_{new})$	5.1	9.59	9.6	9.59	33.88
		π_h	5.1	9.59	9.6	9.59	
		$\bar{c}^*$	0.0	0.0	0.0	0.0	

6.3 State of the Art Algorithm: The Hub Heuristic [57] Algorithm Revisited

This section describes our implementation of the *Hub Heuristic* algorithm, which was proposed in [57]. The corresponding publication was summarized in Section

3.1.3. Algorithm 9 provides a high-level description and essential steps of the *Hub Heuristic*. The algorithm's input is a road network G, and a vehicle set H. Lines 3 - 6 initializes the platoons with exactly one vehicle. For each platoon, the algorithm calculates a vector $(rank(e_1) \dots rank(e_{|E|}))^\intercal$, which contains a normalized edge rank $[0, 1]$ for each edge in the road network G. A detour of the shortest path P_h decreases the rank value, edges with rank value 0 are not suitable for platooning. In case the edge is on the shortest path, the rank value of the edge is set to 1. Lines 7 - 9 determine the platooning pairs with maximum rank and merge them. The rank of two platoons can be calculated by multiplication of their rank vectors and then summing the value in the vector. This process is repeated until no more merging is possible. Lines 10 - 14 calculate the routings for the formed platoons. For each platoon, a hub node $v_{hub} \in V$ with maximum rank is determined. Afterward, the algorithm solves the problem(Eq. 2.8 - Eq. 2.12) from starting nodes of $vehicles \in$ *platoon* to v_{hub} and from v_{hub} to destinations of $vehicles \in$ *platoon*.

Algorithm 9 Hub Heuristic based on [57]

Data: Road network $G = (V, E)$, vehicle set H
Result: (Optimal) platoon routing $O = \{P_1^{pl}, \dots, P_{|H|}^{pl}\}$ for vehicle set H.

1: $O \leftarrow \emptyset$
2: *platoons* $\leftarrow \emptyset$
3: **for** $h \in H$ **do**
4: Calculate shortest path P_h with Dijkstra's algorithm on G.
5: Calculate edge rank $\forall e \in E$ based on P_h.
6: *platoons* $\leftarrow$ Create a platoon with h and associated edge rankings.
7: **while** *stopping condition* **do**
8: Find platooning pair $(platoon_1, platoon_2) \in \binom{platoons}{2}$ with maximal rank.
9: *platoons* = *platoons* $\cup$ Merge $(platoon_1, platoon_2)$.
10: **for** *platoon* $\in$ *platoons* **do**
11: Find a $v_{hub} \in V$ with maximal rank for the *platoon*.
12: Solve the problem from starting nodes of $vehicles \in$ *platoon* to v_{hub}.
13: Solve the problem from v_{hub} to destinations of $vehicles \in$ *platoon*.
14: $O = O \cup$ Combination of two solutions.

6.4 Selecting the Best Mutually Disjoint Groups

Depending on the used grouping algorithm, the group set $\tilde{S}$ can contain non-disjoint vehicle groups. The following approach selects the best disjoint groups from a non-disjoint group set $\tilde{S}$. Eq. 6.21 is based on previously calculated platooning routes and considers group savings $\alpha_{\tilde{C}}$ to find fuel-optimal disjoint groups. The decision variable $x_{\tilde{C}}$ is 1 if given vehicle included in group $\tilde{C}$, otherwise is 0. The problem constraint enforces that a particular vehicle does not occur more than in one group [88].

$$\begin{aligned} \text{maximize} \quad & \sum_{\tilde{C} \in \tilde{S}} x_{\tilde{C}} \cdot \alpha_{\tilde{C}} \\ \text{subject to} \quad & \sum_{\substack{\tilde{C} \in \tilde{S}: \\ h \in \tilde{C}}} x_{\tilde{C}} \leq 1 \quad \forall h \in H \\ & x_{\tilde{C}} \in \{0, 1\} \quad \forall \tilde{C} \in \tilde{S} \end{aligned} \tag{6.21}$$

6.5 Time Scheduling for Platooning Routing

The previous sections described different approaches to solve VPP without taking into account the time restrictions. The primary idea of this work is to create efficient, scalable heuristics independent of the vehicle time restrictions. The computed platoon routing can be used to find solutions for the problem with time restrictions. Such strategies are often used to solve more complex problems with time e.g., the Vehicle Routing Problem with Time Windows (VRPTW) [87, 49]. [93] determines platoons based on a vehicle's shortest paths and afterward calculates the pairwise platoon routing. [74] proposes a genetic algorithm to solve the platooning problem considering of the time restrictions. In experiments, small road networks with 137 nodes and a small number of vehicles are used. [99] formulates a mixed-integer linear problem with soft time constraints. A small road network with 16 nodes and 22 edges is used for the experiments. [72] proposes an ant colony optimization approach to solve the platooning problem. After the determination of the platoon routings, the travel times are calculated. The largest road network used in the experiments consists of 500 nodes with up to 500 vehicles.

This section provides a naive approach to determine the vehicle's departure times for the platooning edges. These edges $e \in \bigcup O$ are included in the platoon routing. The Algorithm 10 constructs a time graph $G_{Time} \subseteq G$ from the platooning routes O. The time graph G_{Time} is initialized by the nodes a_g and b_g, which are connected to the vehicle start nodes a_h and to the vehicle destination nodes b_h. The edges of the platoon routing are added to the time graph with the corresponding travel times, as shown in lines 3 - 9. Lines 10 - 11 calculate the longest paths from a_g to all edges in G_{Time}, and these are the vehicle's departure times.

Algorithm 10 Determine time variables

Data: Road network instance $G = (V, E)$, (optimal) vehicle path set O
Result: Feasible set of times $depart_e \in T \quad \forall e \in E$.

1: $T \leftarrow depart_e = 0 \quad \forall e \in E$
2: *Initialization of time graph G_{Time} by the nodes a_g and b_g*
3: **for** $P_h \in O$ **do**
4: **for** $e = (v, u) \in P_h$ **do**
5: $G_{Time} \leftarrow G_{Time} \cup e$ with $t(e) = \frac{w(e)}{\vartheta}$
6: **if** $v = a_h$ **then**
7: $G_{Time} \leftarrow G_{Time} \cup e = (a_g, a_h)$ with $t(e) = t^a$
8: **if** $u = b_h$ **then**
9: $G_{Time} \leftarrow G_{Time} \cup e = (b_h, b_g)$ with $t(e) = t^b$
10: **for** $e = (v, u) \in G_{Time}$ **do**
11: $depart_e \leftarrow t(P^{lp}_{a_g,v})$

6.6 Summary

This chapter presented different approaches to find optimal or near-optimal platoon routings. Section 6.1 described an *Exact Solver*, which can determine optimal solutions for VPP. In addition, the convergence behavior and scalability were analytically discussed. Section 6.2 introduced our novel CG approach for VPP. A path-based MP was formulated equivalent to ILP. Furthermore, the subproblem was

formulated as a shortest path problem. Section 6.3 described our implementation of the state-of-the-art algorithm *Hub Heuristic* [57]. Section 6.5 illustrated an algorithm to determine the vehicle scheduling for the pre-calculated platoon routings.

Chapter 7

Graph Data Model for Graph Database Support of Platooning

This chapter introduces the data used in the experiments. Testing algorithms require different road networks and vehicle assignment datasets to investigate scalability, quality, and performance. The road networks are imported OpenStreetMap (OSM) [76]. OSM data is free for personal and commercial use under the Open Data Commons Open Database License. The vehicle datasets are synthetically generated, based on the imported road networks. Vehicles are distributed under specific conditions. Furthermore, this chapter describes a new graph data model, which provides a comprehensive view of the road network, vehicle, and platooning data. Finally, Section 7.3 introduces the architecture and implementation of the VPP prototype.

7.1 Transformation of Road Networks from OpenStreetMap to a Graph Database [90]

OSM data is available in XML format, which is not relation-centric organized. OSM's conceptual data model represents only geometric objects and does not allow efficient traversing through these objects [35]. Efficient routing from start points to endpoints is an important prerequisite for the previously introduced algorithms; see Chapter 5 and 6. Our methodology presented in [90] is used for extracting OSM road network data and transforming it into a graph database with graph structure (vertices, edges, and properties) so that the road network data is available in a graph database management system (DBMS) such as Neo4j. We implemented a prototype system *OSM2RN transformer* using Java. *OSM2RN transformer* uses an OSM/XML file as input, filters non-relevant information, and transforms and stores the results into a graph database. Test results are strongly dependent on the size and type of road network. OSM data can be downloaded by country or by sub-region of a country. The filter function of the *OSM2RN transformer* allows us to discard non-relevant information and select the necessary road types. In OSM, the roads with road types *motorway* and *trunk* are always directed and, therefore, tagged as *oneway=yes*. Other roads are represented as directed and/or non-directed geometric objects, see Fig. 7.1 for an example. *OSM2RN transformer* converts the undirected road segments into directed edges so that a directed graph $G = (V, E)$ is created. The granularity of a road network can be controlled by selecting and combining different road network types. Transforming only particular road types can result in a disconnected graph. This problem can be handled by deleting strongly unconnected components. The road network consists of nodes (i.e., road points with coordinates) and edges (i.e., road segments with distance). Additionally, Points of Interest (POI) are specified. In the platooning context, the POI represents depots, the potential start and destination

FIGURE 7.1: Directed or non-directed OSM geometric objects, the red circle shows split from non-directed to directed representation (or vice versa).

points for vehicles. Natively, OSM contains partially non-relevant data, which must be cleaned up.

7.2 Graph Data Model for VPP

This section describes the graph data model for the data-intensive Vehicle Platooning Problem (VPP). Graph databases provide native support for data and relationships between data so that interconnected datasets modeled as a graph structure can be efficiently queried [82, 95]. The VPP graph data model is a flexible labeled property graph model that implements previously presented concepts and corresponding data. Fig. 7.2 depicts the data model, which is described below step-by-step. Road points are modeled as nodes with a label *RoadPoint* and properties *latitude* and *longitude*. The *RoadPoint* entities are connected by directed relationships with *ROAD_SEGMENT* type and *distance_meter* property. The intuitive modeling of the road network as a graph enables the use of graph algorithms. In particular, fast pathfinding can be supported adequately, e.g., efficient determining of the shortest path between two locations with A*. All road points are indexed by R-Tree [39] for improved performance of spatial operations. The road network data can be restricted by specified polygons so that the road area covered by Vehicle Geometric Convex Container (VGCC) can easily be queried. Depots are represented by *Depot* nodes, which are indirectly connected to their respective *RoadPoint*. The *Location* nodes are the so-called *filter node*, which decouples the road network data from other data. Incoming and outgoing relationships of each *RoadPoint* node are stored in a single doubly-linked list, which has to be scanned entirely for each traversal. The *Location* node ensures that the number of relations and the querying performance on road network data remains constant for different numbers of depots, vehicles, and routes. Each *RoadPoint* has exactly one *Location* node, and the additional effort for storing extra location nodes is limited.

Vehicles are modeled as nodes with labels *Vehicle* and *IVehicle*, and have relationship *STARTS_AT* and *ENDS_AT* to the respective *Location* node. Each vehicle belongs exactly to one *VehicleSet* node. This node keeps the vehicle set specific information like the kind of vehicle distribution. Each *VehiclePair* node stores pairwise incentives for exactly two vehicles; this information is most relevant for the grouping process. Formed vehicle groups are modeled as *Group* nodes with relevant group information, e.g., average incentives. Each *Group* node is connected via the *CONSISTS_OF* relationship to the vehicles, which belong to this group. Section 5.3.1 introduced the vehicle assignment partitioning concept, which describes how

to generate new vehicles that represent and replace the original one. Newly generated vehicles are represented by *Partition* nodes, which have the same properties and labels as *Vehicle* and *IVehicle* nodes, i.e., the newly generated vehicle acts like a "standard" vehicle. All partitions are coupled by *COUPLE* relationships so that the original vehicle can be reconstructed in the proper order. The original or partitioned vehicle is converted into a *PartitionedVehicle* node after partitioning. The *PartitionedVehicle* node keeps all original vehicle information and can be distinguished from "normal" vehicles. A list of *RouteNode* nodes represents the final vehicle route. Each vehicle has *STARTS_AT*, *ENDS_AT*, and *IS_ROUTED_BY* relationship to the *RouteNode* nodes. The order of *RouteNode* nodes is defined by the relationship *NEXT* with a distance property. Thus, the calculated vehicle route and the corresponding location nodes can be easily retrieved. Information about formed platoons is kept in *Platoon* nodes. *RouteNode* nodes represents the corresponding platoon route. A single platoon has at least route nodes from two different vehicles.

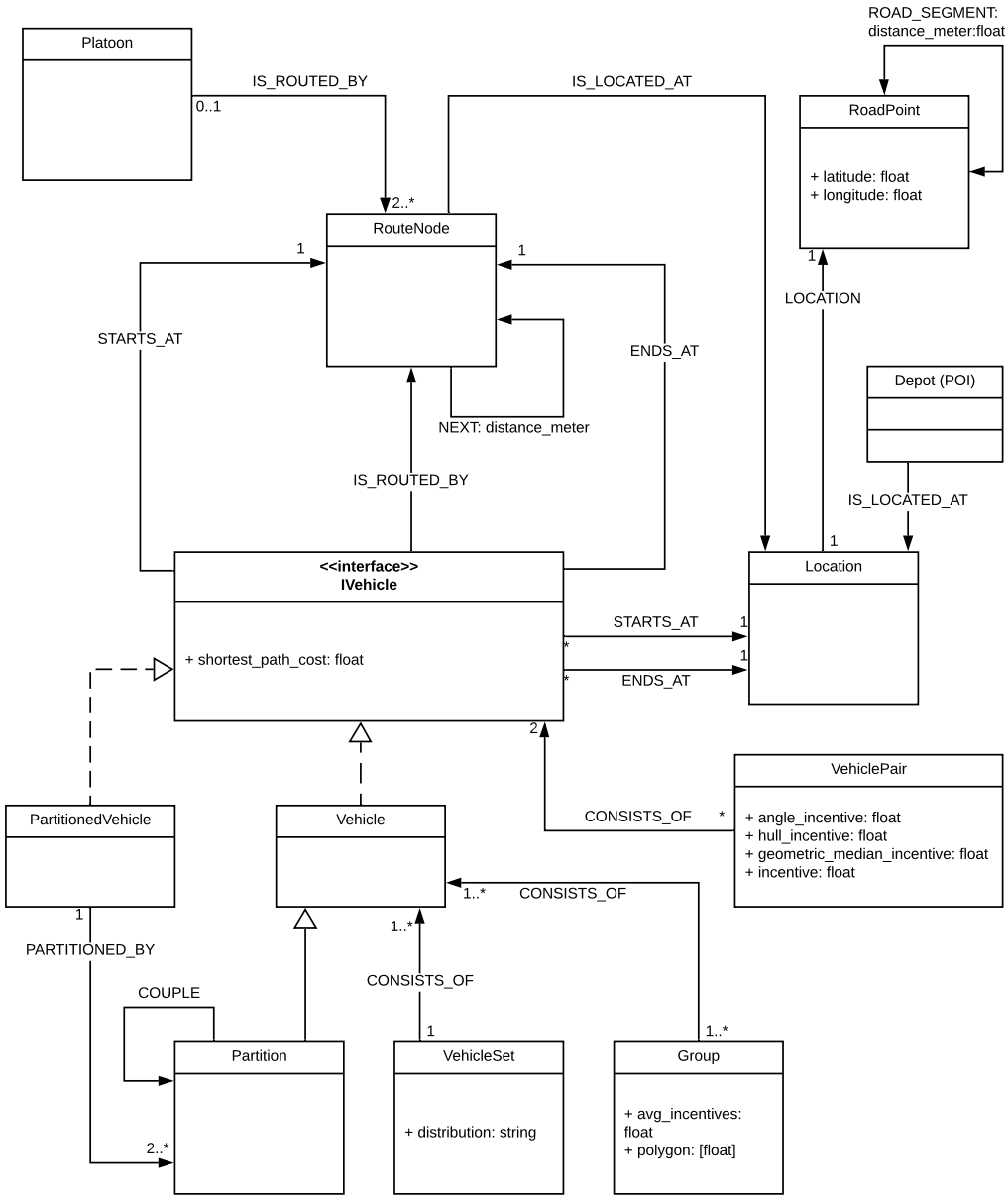


FIGURE 7.2: Graph data model for the VPP.

This graph data model is based on previously proposed algorithms in this thesis and the corresponding requirements. The necessary data for the VPP can be stored and managed efficiently with the proposed graph data model.

7.3 Design of VPP Prototype

This section describes a prototypical implementation of the previously introduced concepts. Our proposed algorithms can be used in different ways, depending on the solution strategy. A promising approach to tackle VPP is to divide it into three subproblems (*Pairwise Vehicle Incentive Ranking*, *Vehicle Grouping*, *and Group-wise Vehicle Routing*) and solve these with different approaches. To wrap different approaches, we created a component-based prototype with dynamically interchangeable components. Experiments can be easily configured with different problem and algorithm characteristics to draw meaningful conclusions. The graph database management system *Neo4j* completely manages algorithm data. For linear optimization, the Gurobi optimizer is used. Section 7.3.1 presents the architecture of our prototype that also meets these requirements.

7.3.1 Prototype Architecture

Fig. 7.3 shows the overall architecture with the crucial components. *Global Configuration* manages environment factors such as *database memory size*, *chunk size*, *output flags*, *number of threads*. A *Controller* is the central component of the system. It takes control of the conduction of our experiments. *TestSuite* constructs a collection of test cases (*TestCase*) based on the experiment configuration (*Experiment Configuration*). The problem characteristics, as well as the algorithm components or parameters, are encoded in JSON format. *TestSuite* creates parameterized algorithm objects from the *Algorithm* component, taking into account the desired combinations. The *Algorithm* component provides a simple interface for the creation and execution of concrete algorithms. For each combination of experimental factors, we get one *TestCase* with a list of algorithms. *TestSuite* runs all *TestCases*, and *TestCase* executes the respective algorithms sequentially. The model component is a database encapsulation layer, which hides the implementation details of the used database. In our realization, the graph database management system *Neo4j* is used [70]. The Experimentum framework component manages experimental outcomes. Our framework Experimentum provides automated support for performing a range of routine tasks, enabling researchers to conduct computational experiments more efficiently. Complex result data from experiments can be managed, analyzed, and visualized conveniently. We have implemented our framework as a prototype and tested it with various application cases from different application domains [30, 79].

The entire prototype is implemented using the programming language Python 3.6.*. Furthermore, several data science packages such as NumPy, Pandas, SciPy, Scikit-learn, and seaborn are used for efficient in-memory data processing. Previously formulated. The Integer Linear Program is realized using the Gurobi Optimizer 8 [64].

7.3.2 Memory Management and Optimization

Handling extensive problem instances requires careful memory management. The biggest challenge is the allocation of the available memory. Our prototype's most significant memory consumers are the graph database management system Neo4j

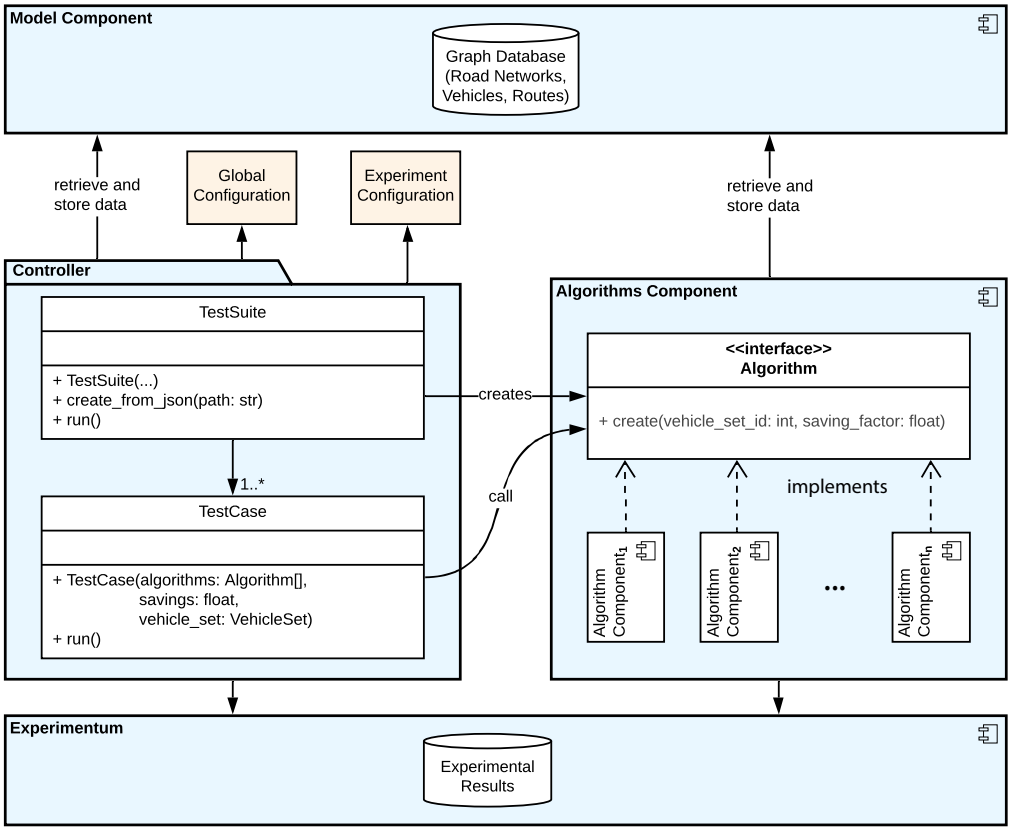


FIGURE 7.3: Architecture of VPP prototype.

and the Gurobi Optimizer. Certain graph database queries are very complex, and the explicit planning of transactions is necessary. One of the data-intensive tasks is the management of vehicle incentives. For example, for 2000 vehicles, 1998000 incentives are calculated, and 1998000 nodes and 3996000 relationships must be stored. Our system distributes such queries uniformly over a configurable number of transactions. Depending on the available memory, the size of the transactions can be adjusted. Furthermore, Neo4j provides built-in procedures for periodic executions [69]. This function is used to create a more significant number of vehicle shortest paths and to clean up the data after each execution of the test cases. The memory usage in Neo4j can be configured in different ways, see [68]. For our purposes, the setting of the heap space is especially important. The memory size for Neo4j has to be balanced, because the Gurobi Optimizer also consumes much memory, especially for larger problem instances. The Gurobi Optimizer provides many possibilities to configure memory usage [38]. The *NodefileStart* parameter specifies a memory threshold for storing the Mixed Integer Programming (MIP) tree nodes to disk. In our case, the parameter is set to 0.1, to release the memory. The *Threads* parameter defines the number of parallel threads. By default, all cores are used. This parameter is set to 1 so that only one core is used to reduce memory usage. Using more threads in our case slows down the solution process.

Chapter 8

Experimental Evaluation

This chapter compares the performance of our proposed algorithms in Chapter 5 and Chapter 6. Section 8.1 describes the setup of experiments, problem instances, and used technologies. In Section 8.2, we evaluate grouping algorithms with the ILP routing method. In Section 8.3, we evaluate routing algorithms in comparison to the state-of-the-art algorithms from the related work. Section 8.4 evaluates the composition of grouping and routing algorithms with the largest road network instance.

8.1 Defining the Experiments

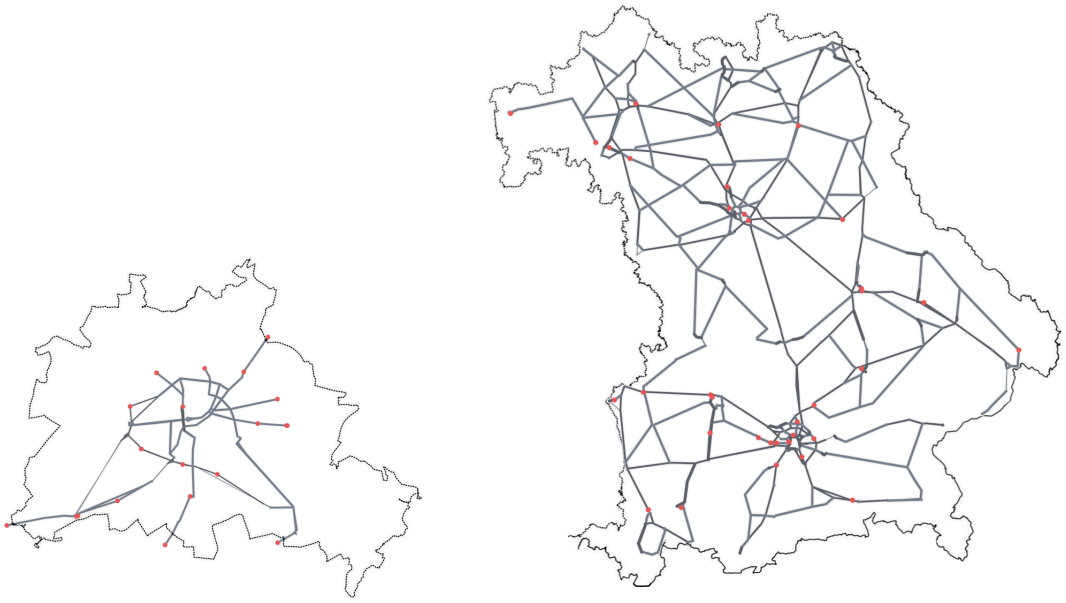
We use artificially generated grid road networks and real-world road networks to test performance and scalability under controlled conditions for the experiments. The real-world road networks are generated from OSM data and imported into the graph database by *OSM2RN transformer*. Different sizes of road networks have been chosen to test the scalability of all algorithms. Table 8.1 shows an overview of used road networks. The grid road networks are regular square grids with 10×10, 20×20, and 30×30 nodes, also called 90-degree networks. The edges have a fixed distance of $c(e) = 50$ km. The grid networks contain generated a POI, i.e., depots. These are randomly assigned to the *RoadPoints*. In addition to the synthetic grid road networks, we also use real road networks with different topologies and sizes. The depots are real DHL facilities [29, 24] or Swedish postal service centers, respectively.

TABLE 8.1: Imported road networks from OSM and artificially generated grid road networks.

Road Networks	Nodes	Edges	Depots	Figures
Grid 10 $(50km/e)$	100	360	10	
Grid 20 $(50km/e)$	400	1520	20	
Grid 30 $(50km/e)$	900	3480	30	
Berlin	298	806	20	Fig. 8.1a
Bavaria	3739	7474	36	Fig. 8.1b
Sweden	6668	15748	155	Fig. 8.4a
Germany	31132	66173	257	Fig. 8.4b
The Southern United States (17 States)	133174	280551	1568	Fig. 8.31

The complexity of road networks increases by the size and topology of the road networks. The real road networks contain different topological patterns. Some areas have high density, e.g., a metropolis with excellent motorway connections.

The size of the vehicle set has a significant impact on computation time. We have generated vehicle sets of varying sizes $[10, 20, 30, 50, 100, 200, 300]$ to investigate the



(a) The Berlin Road Network with 298 nodes and 806 edges.

(b) The Bavarian Road Network with 3739 nodes and 7474 edges.

FIGURE 8.1: The road network are generated from OSM data with [90]. The depots are illustrated as red points.

scalability. The start nodes a and destination nodes b of vehicle assignments are placed on depots to get closer to the real-world situation. All vehicle sets are persistent so that the experiments can be repeated at any time. All test cases have been repeated ten times. The experiments have been performed on two Linux Ubuntu 18.04 servers with AMD Ryzen 7 1800X Eight-Core Processor 3.60 GHz and 65.96 gigabytes DDR4. The business intelligence Tool Tableau is used to visualize the road network [15, 32]. Experimental data are processed by python data analysis library *pandas* [78] and visualized by *Matplotlib* and *seaborn* [63, 84].

8.1.1 How are the experimental results presented?

The experimental results are divided into Section 8.2, Section 8.3, and Section 8.4. Each section contains **subsections** for each **road network instance**. For each road network instance, the results are illustrated with **a pair of diagrams** for the different vehicle sets, such as in Fig. 8.2 and Fig. 8.3. **Line plots** represent the **execution times** of the algorithms, as shown, for example, in Fig. 8.2. The y-axis represents the execution time in seconds, and the x-axis represents the vehicle sets according to their size. The corresponding **bar plot** represents the percentage of achieved **fuel savings**, and an example of this is shown in Fig. 8.3. The fuel savings are grouped by the vehicle sets.

The **relative fuel savings** can indicate the quality of a platoon routing. The **relative fuel savings** θ are expressed as a percentage of fuel savings achieved by platoon routing O compared to the shortest paths $c(P_h) \ \ \forall h \in H$; see Eq. 8.1.

$$\theta = \frac{\sum\limits_{h \in H} c(P_h) - c(O)}{\sum\limits_{h \in H} c(P_h)} \cdot 100, \quad O \subseteq (E \times \mathbb{Z}_+)^{|H|} \tag{8.1}$$

8.2 Computational Experiments of Grouping-based Routing Algorithms

In Chapter 5, we proposed several heuristic algorithms for the formation of vehicle groups. This section investigates the performance of grouping-based routing algorithms compared to the exact solving of the VPP, which is formulated as the ILP model in Section 2.1.1 (*Exact Solver*). In other words, we compute pairwise incentives (described in Section 5.1) with the standard parameters. The parameters are specified in Section 8.5. Based on the calculated incentives, the groups are formed with the α-*cut Method*, *Exact CPP*, *Greedy CPP*, or *SGVNS CPP* method. The routing method (Section 2.1.1) calculates the platooning routing group-wise, see Algorithm 1. As a baseline, we also compute the optimal solution without vehicle grouping with the *Exact Solver*.

10×10 Grid Road Network Instance

Fig. 8.2 and Fig. 8.3 presents the performance of α-*cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* (inclusive of incentives calculation time and group-wise platooning routing calculation time) on the 10×10 grid road network with 10, 20, 30 and 50 vehicle instances. For the instance with ten vehicles, the execution time on the median of all grouping algorithms and the *Exact Solver* is less than one second. The relative savings of grouping algorithms are between 0.96% and 1.76%, where the optimal savings are 2.24%. For the instance with 20 vehicles, the execution time of grouping-based routing algorithms is worse than the execution time of the *Exact Solver*. Savings also increase with the number of vehicles to up to 4.90%. The α-*cut Method* and *Greedy CPP* achieve 24.48% and 11.63% fewer savings than other algorithms. For the instances with 30 and 50 vehicles, the time gap between grouping-based routing algorithms and *Exact Solver* becomes significantly larger.

For the instance with 30 vehicles, the α-*cut Method* and *Greedy CPP* approaches reach 4.16% and 4.69% savings. For the instance with 50 vehicles, the obtained solutions of α-*cut Method* and *Greedy CPP* deliver 4.51% and 5.35% savings. For the instances with 30 and 50 vehicles, the *Exact CPP* and *SGVNS CPP* methods almost achieve the optimal solution in terms of savings. The execution time of grouping-based routing algorithms grows relative to the number of vehicles. The pairwise calculations of the incentives take the most time. The execution time of the *Exact Solver* remains nearly constant.

Fig. 8.5 and Fig. 8.6 shows the performance for 100, 200, and 300 vehicles on the 10×10 grid road network. The plot in Fig. 8.5 uses a logarithmic scale for the time representation, due to the broad range of quantities. The execution time of *Exact CPP* has grown drastically, compared to the 50 vehicles instance, the 100 vehicles instance is 44 times slower. Formation of groups with *Exact CPP*, see Eq. 5.20, takes $\approx 90\%$ of the calculation time. Table 8.2 provides a detailed report of the execution times for the 10×10 grid road network instance. The grouping-based routing algorithms are compared to the baseline *Exact Solver* approach. The *overhead* columns represent the additional computational time compared to the *Exact Solver* approach.

For the instance with 100 vehicles, the *Exact CPP* and *SGVNS CPP* achieve slightly fewer savings with 7.16% than the optimal savings with $\theta^* = 7.65\%$. The *Greedy CPP* approach achieves 6.76%, and considerably fewer reaches the α-*cut Method* method with 6.09%. For the instance with 200 vehicles, the execution time of the *Exact CPP* approach explodes to 11108 seconds ≈ 3 hours. The majority of time is used solving the linear program, which forms the groups; see Eq. 5.20. The

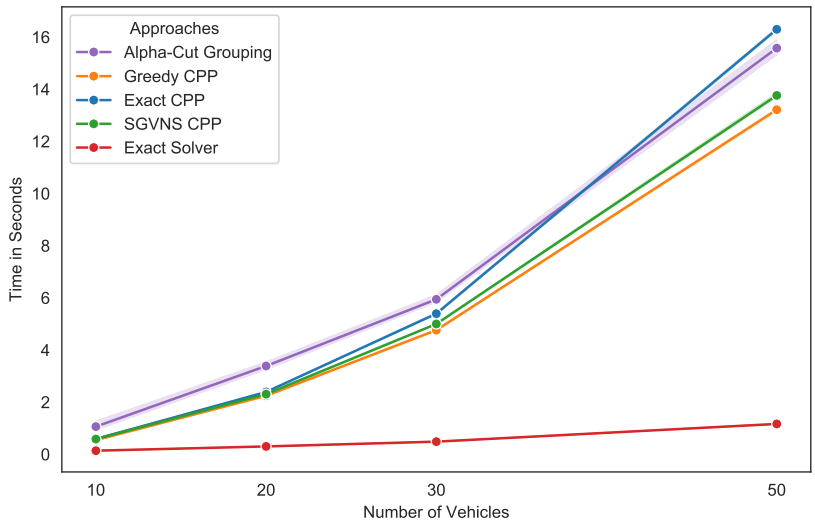


FIGURE 8.2: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 10, 20, 30, and 50 vehicles on the 10×10 grid road network.

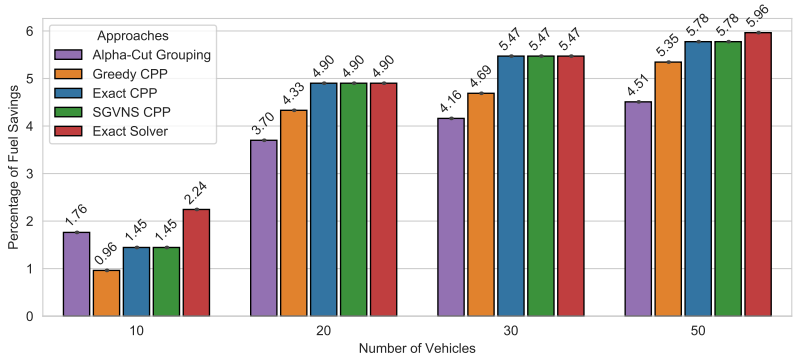
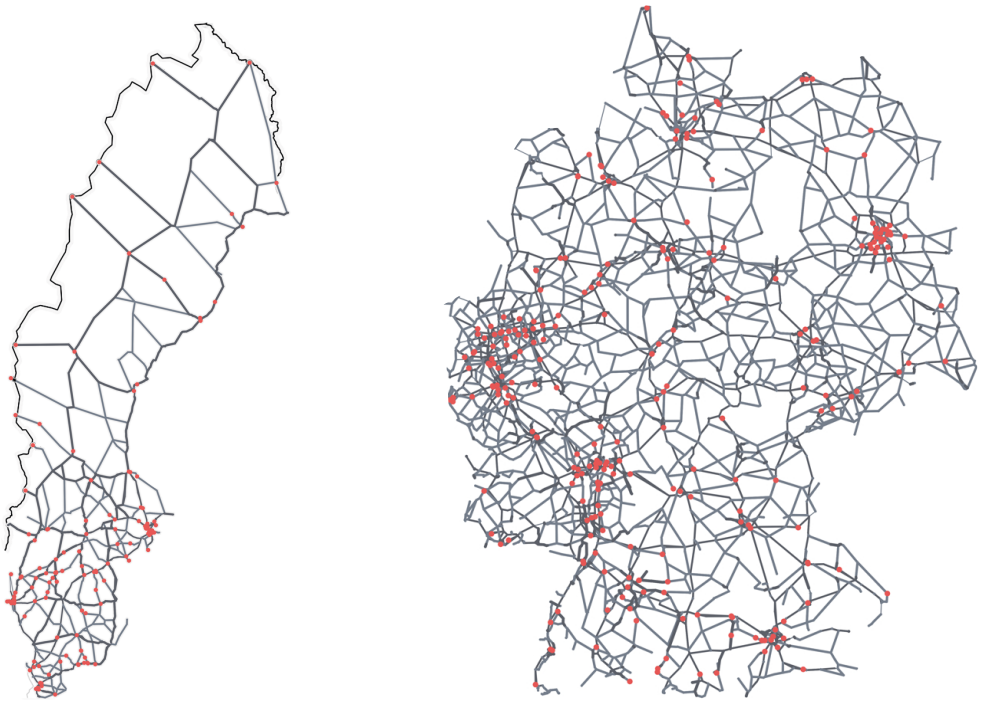


FIGURE 8.3: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 10×10 grid road network.

time limit of the Gurobi Optimizer is set to 10800 seconds. Without a time limit, the computation takes up to 50000 seconds $\approx$ 13 hours. In this case, the *SGVNS CPP* approach achieves more savings than the *Exact CPP* approach with 8.04% vs. 7.95%. By stopping early, the *Exact CPP* no longer determines the optimal groups, and therefore fewer savings are determined. The execution time of the *α-cut Method*, *Greedy CPP*, and *SGVNS CPP* approaches are very similar. *α-cut Method* reaches 7.78% savings with 400 groups. The *SGVNS CPP* approach achieves 8.04% savings, the distance to the optimal fuel-savings is 0.38%.

For the instance with 300 vehicles, the execution time of the *Exact CPP* reaches the time limit and achieves the near-optimal solution with 8.46%. The *SGVNS CPP*, *Greedy CPP*, and *α-cut Method* achieve 8.32%, 8.44%, and 8.33% savings. The optimal solution (8.78% savings) is determined in ≈ 8 seconds.



(a) The Swedish Road Network with 6668 nodes and 15748 edges.

(b) The German road network with 31132 nodes and 66173 edges.

FIGURE 8.4: Larger road network instances generated from OSM with [90]. The depots are illustrated as red points.

For all vehicle instances, the optimal solution can be determined most efficiently. The Gurobi Solver needs significantly less time to determine the optimal solution than the grouping-based routing algorithms for the (approximated) solutions. The incentives calculation caused longer execution times. The computation time of incentives is constant, which depends mainly on the number of vehicles, see Table 8.10. The calculations can easily be improved by parallel implementation. The execution time of grouping algorithms is mainly constant; only the *Exact CPP* scales poorly for large numbers of vehicles, see Table 8.11. The scalability of the *Exact CPP* also depends on incentive values. *SGVNS CPP* and *Exact CPP* approaches achieve an optimal or near-optimal solution for 20, 30, and 50 vehicles instances. α-*cut Method* approach provides the worst solution of all algorithms in most cases, and the solution quality can be adjusted by $\tilde{k}$ parameter, see Section 8.5.3. It should be noted that the grouping approaches become very important for the more significant problem instances because the *Exact Solver* scalability is very poor.

TABLE 8.2: Execution times of grouping-based routing algorithms in comparison to the baseline on the 10×10 Grid Road Network.

	α-cut Method		*Greedy CPP*		*Exact CPP*		*SGVNS CPP*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	1.06295	664.237	0.547914	293.936	0.588635	323.213	0.5856	321.032	0.139087
20	3.38658	1018.45	2.24465	641.319	2.39006	689.342	2.30764	662.122	0.302791
30	5.94319	1123.28	4.76254	880.268	5.39337	1010.11	4.99871	928.878	0.48584
50	15.5755	1236.67	13.2144	1034.05	16.2969	1298.59	13.7612	1080.98	1.16524
100	61.5372	3458.48	54.9141	3075.49	708.34	40860.8	58.0502	3256.84	1.72931
200	227.69	5135.2	216.174	4870.4	11111.7	255387	223.283	5033.87	4.34922
300	494.53	5791.29	480.313	5621.93	11617.4	138297	502.053	5880.92	8.39425

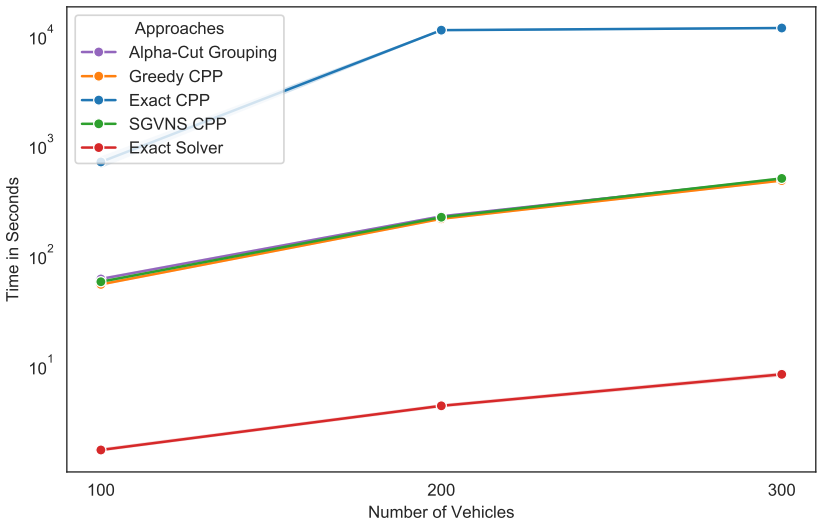


FIGURE 8.5: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 100, 200, and 300 vehicles on the 10×10 grid road network.

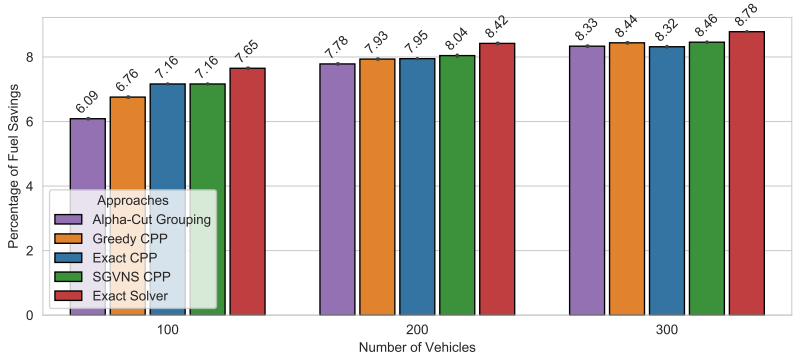


FIGURE 8.6: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 10×10 grid road network.

20×20 Grid Road Network Instance

Fig. 8.7 and Fig. 8.8 describes the performance of grouping algorithms compared to the *Exact Solver* for the 20×20 grid road network. Obtained algorithms have no essential differences in terms of execution time for the instance with ten vehicles. The grouping-based routing algorithms, *Exact CPP* and *SGVNS CPP*, achieve the optimal solution with 2.56% savings.

As in the instance with 20 vehicles, the *Exact CPP* and *SGVNS CPP* approaches almost reach the optimal solution with 4.45% savings. *Greedy CPP* and *α-cut Method* approaches achieve only 3.72% and 2.43% savings. As in the instance with 30 vehicles, the *Exact Solver*, *Exact CPP*, and *SGVNS CPP* achieve 5.5% fuel savings. *Greedy CPP* and *α-cut Method* approaches achieve 4.85% and 3.11% savings. In the instance with 50 vehicles, the execution time substantially increases when compared to the

30 vehicles instance. Furthermore, more substantial differences between the execution times can be observed for the instance with 50 vehicles. The *Exact Solver* has no essential difference compared to *Exact CPP* and *SGVNS CPP* approaches, in terms of performance. The execution time of the *Exact Solver* has deteriorated because the road network is larger.

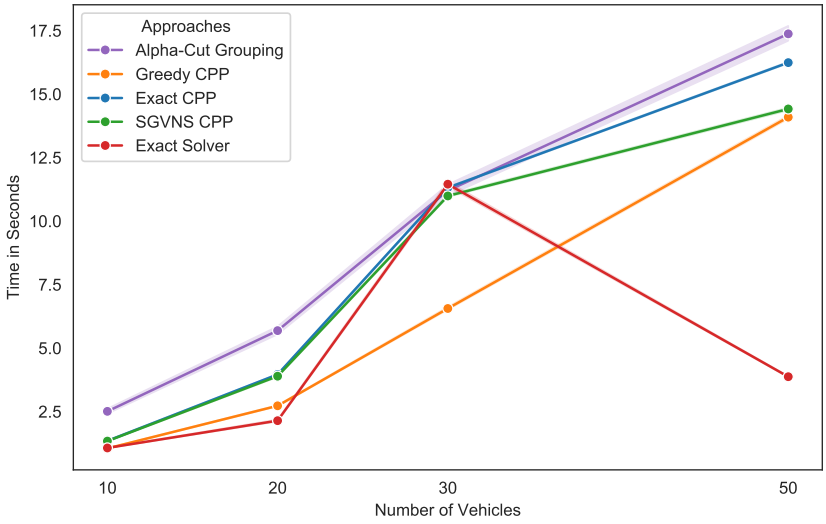


FIGURE 8.7: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 10, 20, 30, and 50 vehicles on the 20×20 grid road network.

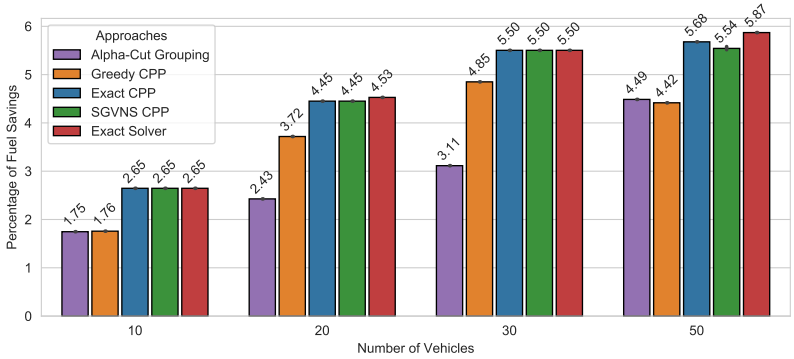


FIGURE 8.8: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 20×20 grid road network.

Figures 8.9 and 8.10 show the performance of the grouping-based routing algorithms compared to the *Exact Solver*. For the comparison, the 100, 200, and 300 vehicle instances on the road network 20×20 grid are used.

In the instance with 100 vehicles, the grouping-based routing algorithms need ten times longer than the *Exact Solver*. The *SGVNS CPP* and *Exact CPP* approaches almost reach the optimal solution with 6.94% and 6.96% savings. As in most cases, the *Greedy CPP* and *α-cut Method* achieve worse solutions with 6.92% and 6.51% savings. In the instance with 200 vehicles, the grouping-based routing algorithms

need significantly more time to solve the instance, because of the incentive calculation, which takes up to 99% of the total execution time. *SGVNS CPP*, *Exact CPP*, *Greedy CPP*, and *α-cut Method* achieve high-quality solutions with 8.44%, 8.42%, 8.44%, and 8.08% savings. For the instance with 300 vehicles, the execution time of *Exact CPP* reaches the time limit. *SGVNS CPP*, *Exact CPP*, and *Greedy CPP* algorithms require up to ≈ 500 seconds to determine the heuristic solution. *Exact CPP*, *SGVNS CPP*, and *Greedy CPP* deliver the near-optimal solution with 8.93%, 8.99%, and 8.99% savings. *α-cut Method* produces 600 small size non-disjoint groups with $\theta^{\alpha-cut} = 8.72\%$ savings. The distribution of the groups is as follows: 93 groups with two vehicles each, 243 groups with three vehicles each, 204 with four vehicles each, and 60 groups with five vehicles each. The *α-cut Method* first creates small groups and then enlarges them iteratively until the group limit is reached. Small groups could harm the solution quality. *Exact CPP* method produces solution with 11 large groups $[30, 33, 18, 30, 22, 46, 29, 57, 17, 9, 9]$.

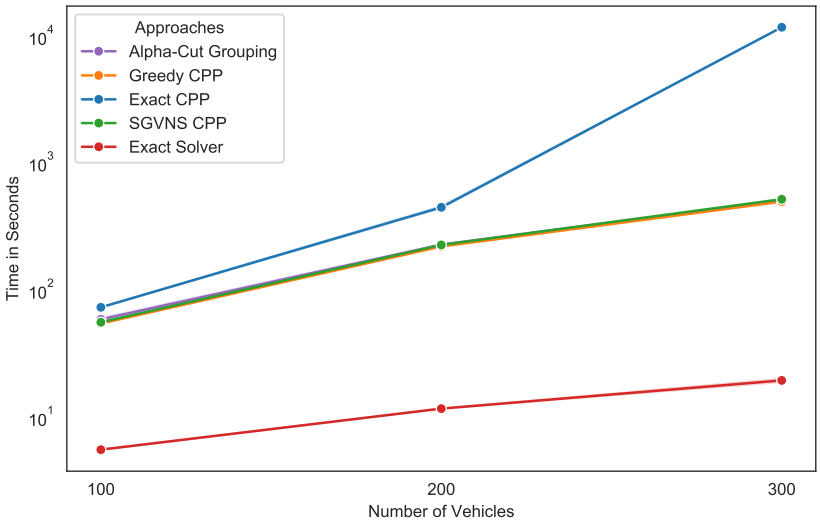


FIGURE 8.9: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 100, 200, and 300 vehicles on the 20×20 grid road network.

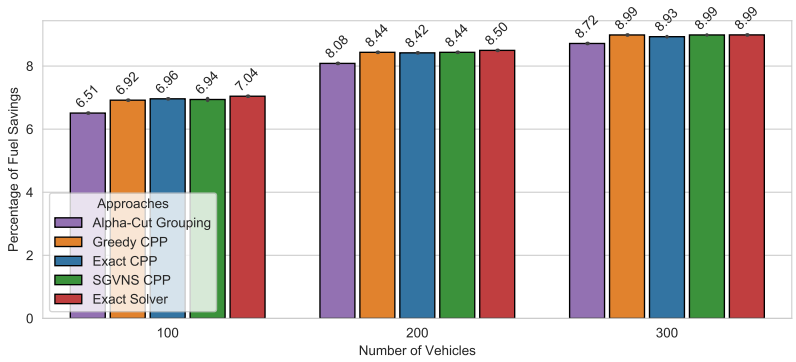


FIGURE 8.10: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 20×20 grid road network.

TABLE 8.3: Execution times of grouping-based routing algorithms in comparison to baseline on the 20×20 Grid Road Network.

	α-cut Method		*Greedy CPP*		*Exact CPP*		*SGVNS CPP*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	2.51066	134.777	1.05094	-1.72483	1.34673	25.9352	1.33952	25.2611	1.06938
20	5.68958	165.429	2.72897	27.311	3.96348	84.903	3.89423	81.6725	2.14354
30	11.2007	-2.22074	6.56076	-42.7262	11.3178	-1.19794	10.9914	-4.04753	11.4551
50	17.374	348.394	14.0929	263.715	16.241	319.152	14.4169	272.076	3.87472
100	59.0402	957.604	54.7848	881.375	73.3309	1213.6	55.9793	902.774	5.58245
200	227.601	1839.1	220.722	1780.49	447.771	3714.89	226.685	1831.3	11.7375
300	506.85	2492.48	496.531	2439.7	11635	59411.8	518.267	2550.88	19.5508

Berlin Road Network Instance

Fig. 8.11 and Fig. 8.12 represents the performance obtained by our grouping algorithms and the *Exact Solver* on the Berlin road network. Ten, 20, 30, and 50 vehicles instances are used for the evaluation. Mentioned execution times in Fig. 8.11 and in Fig. 8.2 for the 10×10 grid road network, show similar patterns. In both cases, the *Exact Solver* is essentially faster (under 2 seconds) than the grouping algorithms. *SGVNS CPP* and *Exact CPP* solve all vehicle instances optimally, and the execution time increases continuously up to 24 seconds. The *α-cut Method* provides near-optimal solution values $\theta_{10}^{\alpha-cut} = 3.21\%$, $\theta_{20}^{\alpha-cut} = 3.76\%$, $\theta_{30}^{\alpha-cut} = 6.34\%$, and $\theta_{50}^{\alpha-cut} = 6.9\%$ savings. The *Greedy CPP* approach delivers similar solutions with $\theta_{10}^{greedy} = 3.76\%$, $\theta_{20}^{greedy} = 3.69\%$, $\theta_{30}^{greedy} = 6.23\%$, and $\theta_{50}^{greedy} = 6.85\%$ savings.

Fig. 8.13 and Fig. 8.14 shows the execution times and achieved savings obtained by our grouping algorithms on the Berlin road network. The execution times are comparable with the execution times in Fig. 8.9. The *Exact Solver* takes 4.2 seconds to solve 100 vehicles instance optimally, while the grouping algorithms need up to 140 seconds to find approximate solutions. *Exact CPP*, *SGVNS CPP*, *Greedy CPP*, and *α-cut Method* deliver near-optimal solutions with 8.32%, 8.31%, 8.34%, and 8.17% savings. To solve the instance with 200 vehicles, the *Exact CPP*, *SGVNS CPP*, and *Greedy CPP* need on average 772.92, 263.4, and 257.53 seconds with $\theta^{optcpp} = \theta^{sgnvs} = \theta^{greedy} = 9.13\%$. In the instance with 300, the *Exact CPP* terminates within the time limit, 10800 seconds. The results show that solving larger CPP instances is computationally expensive. Therefore, efficient heuristics are necessary to determine high-quality solutions, namely disjoint groups with maximum total incentives. Proposed heuristics *SGVNS CPP*, *Greedy CPP*, and *α-cut Method* take ≈ 22, ≈ 3, and ≈ 5 seconds for the same problem instance. We can conclude that the *Exact Solver* has the best performance for considered test cases, except for problem instances on 20×20 grid road network and with 10, 20, 30 and 50 vehicles.

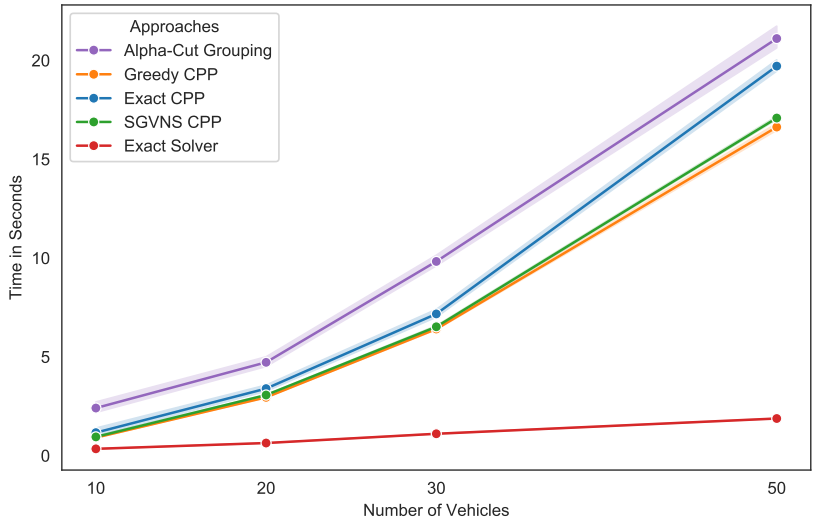


FIGURE 8.11: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 10, 20, 30 and 50 vehicles on the Berlin road network.

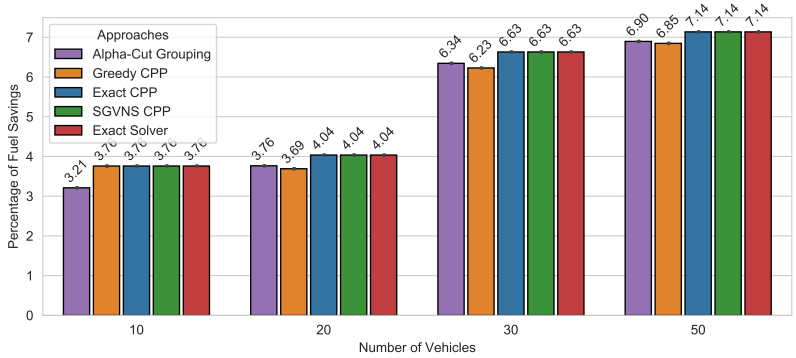


FIGURE 8.12: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Berlin road network.

TABLE 8.4: Execution times of grouping-based routing algorithms in comparison to baseline on the Berlin Road Network.

	α-cut Method		*Greedy CPP*		*Exact CPP*		*SGVNS CPP*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	2.40932	596.896	0.913342	164.185	1.16581	237.212	0.94933	174.594	0.345721
20	4.72306	638.696	2.94991	361.372	3.39405	430.837	3.06819	379.872	0.639378
30	9.82756	787.162	6.40868	478.529	7.17249	547.481	6.52171	488.733	1.10775
50	21.1069	1022.6	16.6266	784.315	19.7122	948.425	17.0841	808.646	1.88017
100	78.4042	1851.42	66.4862	1554.79	139.321	3367.59	67.507	1580.2	4.0178
200	285.693	3210.2	257.753	2886.47	766.881	8785.52	263.476	2952.78	8.63068
300	554.41	4024.87	558.007	4051.63	11868.9	88205.7	580.973	4222.51	13.4407

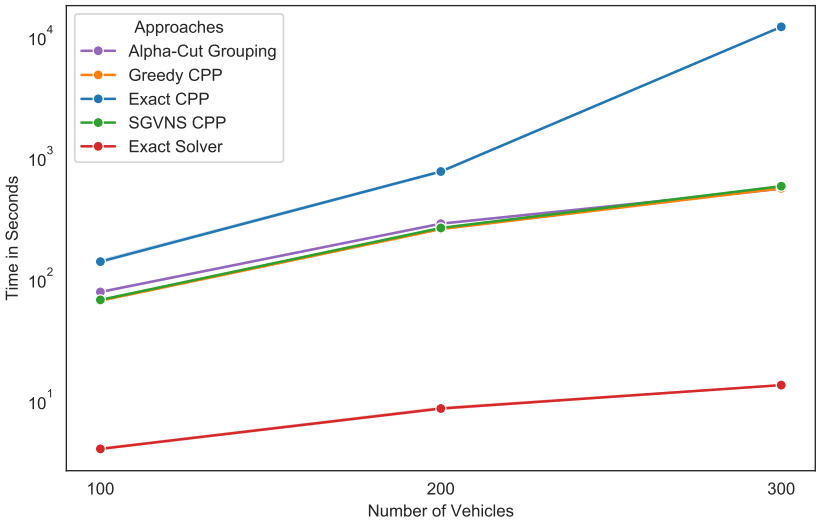


FIGURE 8.13: Execution times of the grouping-based routing algorithms: α-*cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 100, 200, and 300 vehicles on the Berlin road network.

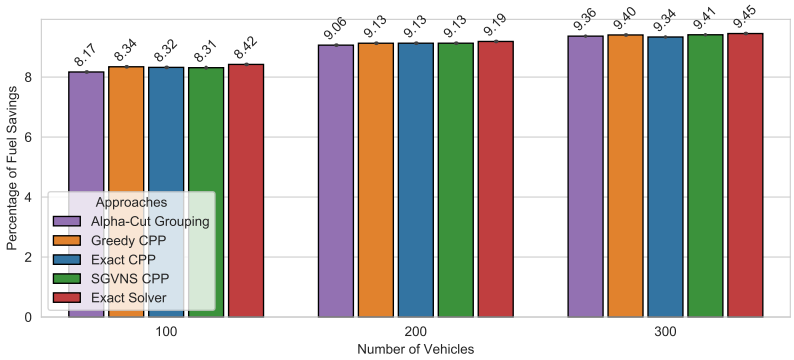


FIGURE 8.14: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 100, 200 and 300 vehicles on the Berlin road network.

Bavarian Road Network Instance

Fig. 8.15 and Fig. 8.16 reports the results for test cases conducted on the Bavarian road network with 3739 nodes and 7474 edges. The execution times show similar trends, especially the execution time of the *Exact Solver* has increased with the size of the road network. The *Exact Solver* takes 5.22 seconds to optimally solve the instance with ten vehicles. Compared with the Berlin road network, the *Exact Solver* solves the 50 vehicles only in 1.9 seconds. For the instances with 20, 30, and 50 vehicles, the execution time increases continuously, as well as for the *Exact Solver* as for the grouping algorithms. Solving the 50 vehicles instance on the Bavarian road network with the *Exact Solver* is **29 times slower** than on the Berlin road network. In this context, the execution time of grouping algorithms has increased by a factor of three. *Exact CPP* and *SGVNS CPP* achieve optimal or near-optimal solutions for all instances.

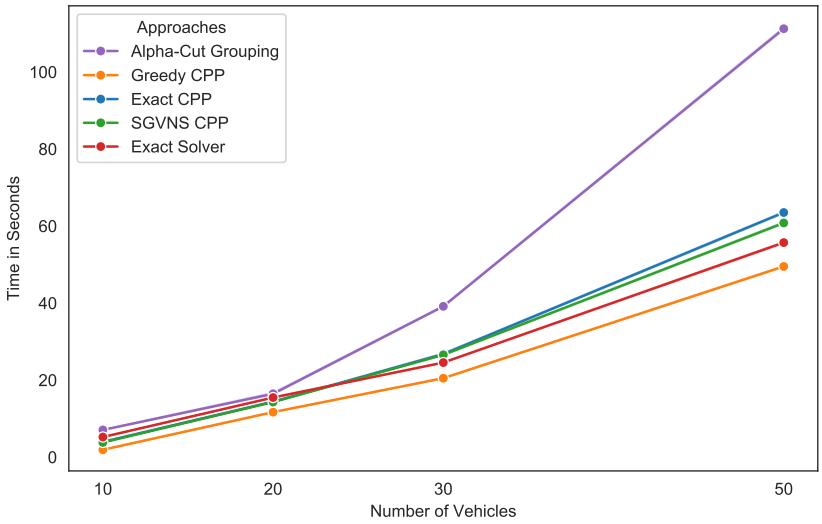


FIGURE 8.15: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 10, 20, 30, and 50 vehicles on the Bavarian road network.

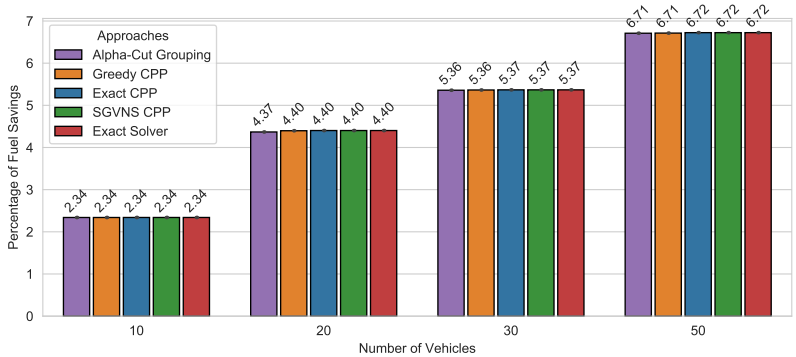


FIGURE 8.16: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Bavarian road network.

Fig. 8.17 and Fig. 8.18 report the performance of our grouping algorithms compared to the *Exact Solver* on the Bavarian road network. The execution time of the *Exact Solver* increases by 17-fold for the instance with 100 vehicles compared to test cases in Fig. 8.13. For the instances with 200 and 300 vehicles, the same behavior can be observed. All proposed grouping algorithms achieve high-quality solutions with a small deviation from the optimal solution, and the *Exact CPP* reaches the time limit.

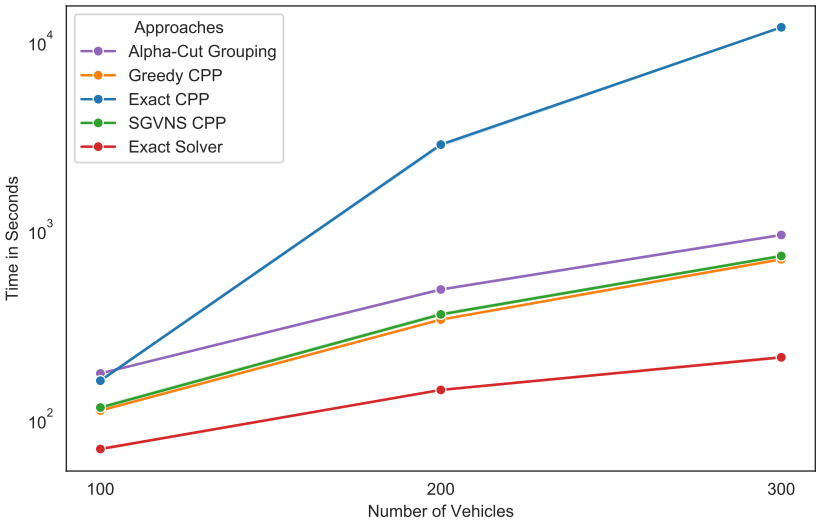


FIGURE 8.17: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 100, 200, and 300 vehicles on the Bavarian road network.

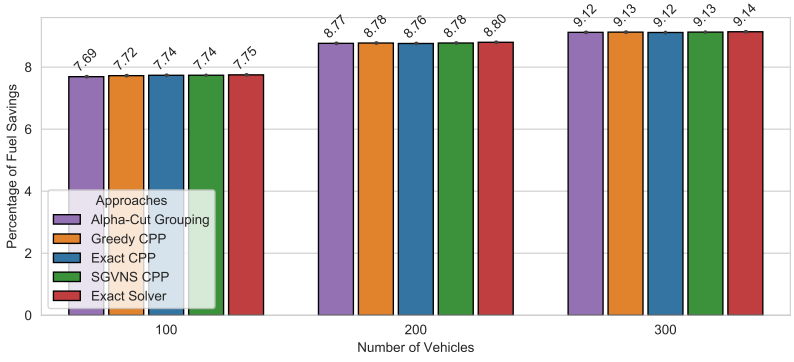


FIGURE 8.18: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Bavarian road network.

TABLE 8.5: Execution times of grouping-based routing algorithms in comparison to baseline on the Bavarian Road Network.

	α-cut Method		*Greedy CPP*		*Exact CPP*		*SGVNS CPP*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	7.0629	35.2719	1.91512	-63.3208	3.95333	-24.2841	3.81186	-26.9935	5.22126
20	16.458	6.36724	11.6859	-24.4746	14.3949	-6.96668	14.3044	-7.55156	15.4728
30	39.1555	59.4707	20.5002	-16.5076	26.7694	9.02521	26.5606	8.17453	24.5534
50	111.24	99.665	49.5085	-11.1366	63.5177	14.0085	60.8062	9.1417	55.7131
100	176.084	150.73	112.087	59.6026	161.228	129.576	116.277	65.5692	70.2287
200	488.418	239.074	338.817	135.216	2843.28	1873.89	360.918	150.56	144.045
300	948.078	342.774	705.192	229.34	11838.2	5428.69	734.274	242.922	214.123

Swedish Road Network Instance

The previously observed trend in the Bavarian road network becomes even more apparent for the road network of Sweden in Fig. 8.19 and Fig. 8.20. The *Exact Solver* needs significantly more time then the *SGVNS CPP*, *Exact CPP*, and *Greedy CPP* to determine high-quality solutions. The execution time of *α-cut Method* shows no significant differences to the *Exact Solver*, tested with Friedman procedure and Bergmann-Hommel post hoc procedure. The complexity of VPP, formulated in (Eq. 2.8 - Eq. 2.12), is strongly dependent on the number of vehicles $|H|$ and size of the road network $|V|$. The road network of Sweden contains twice as many nodes and edges as the Bavarian road network; therefore, the increase in execution times is foreseeable. All grouping algorithms achieve optimal or near-optimal solutions.

TABLE 8.6: Execution times of grouping-based routing algorithms in comparison to baseline on the Swedish Road Network.

	α-cut Method		*Greedy CPP*		*Exact CPP*		*SGVNS CPP*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	12.1536	3.82514	3.44192	-70.5965	8.64032	-26.1878	8.59766	-26.5522	11.7058
20	22.0372	-22.6574	8.57824	-69.8934	10.5238	-63.0652	10.4275	-63.4031	28.4929
30	33.9861	-20.9212	12.1665	-71.6909	15.6362	-63.6176	16.1946	-62.3184	42.9775
50	87.5053	-13.1335	33.7293	-66.517	52.0709	-48.3092	49.9655	-50.3992	100.735
100	392.894	3.12903	405.014	6.3104	3041.11	698.246	448.986	17.8522	380.974
200	628.042	42.6829	510.409	15.9582	11370.7	2483.27	518.101	17.7058	440.166
300	925.147	113.55	682.423	57.5225	11825.9	2629.74	705.02	62.7386	433.223

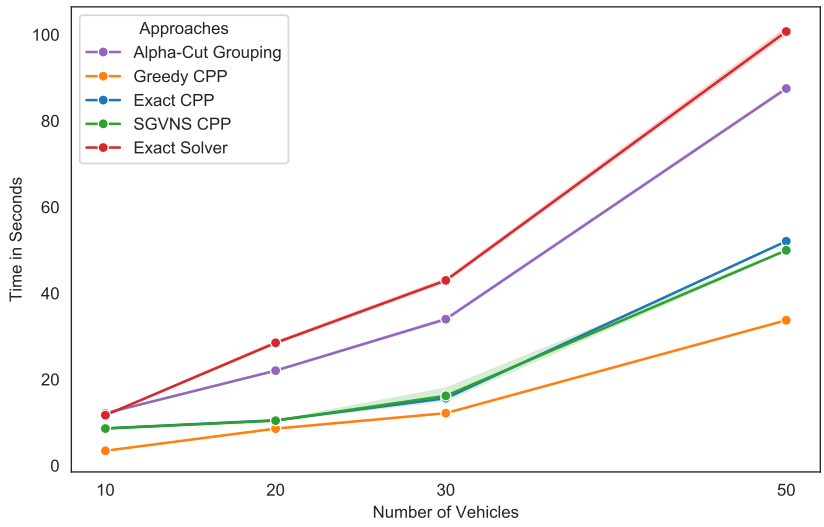


FIGURE 8.19: Execution times of the grouping-based routing algorithms: α-*cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 10, 20, 30 and 50 vehicles on the Swedish road network.

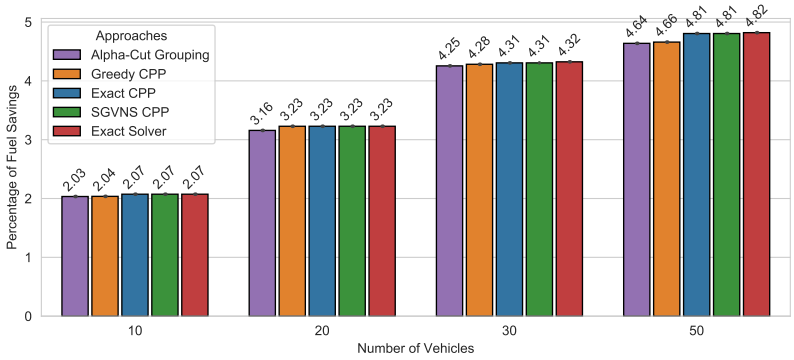


FIGURE 8.20: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Swedish road network.

Fig. 8.21 and Fig. 8.22 shows the performance of algorithms on the Swedish road network. *SGVNS CPP*, *Greedy CPP*, and α-*cut Method* have no significant differences to the *Exact Solver* in terms of execution time. *Exact CPP*, *SGVNS CPP*, *Greedy CPP*, and α-*cut Method* algorithms achieve near-optimal solutions for all vehicle instances. The *Exact CPP* approach reaches the time limit for the instances with 200 and 300 vehicles.

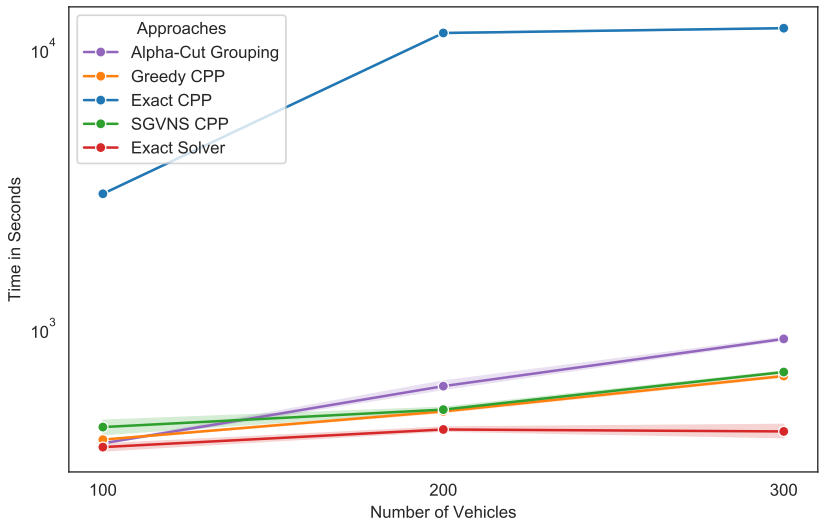


FIGURE 8.21: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 100, 200, and 300 vehicles on the Swedish road network.

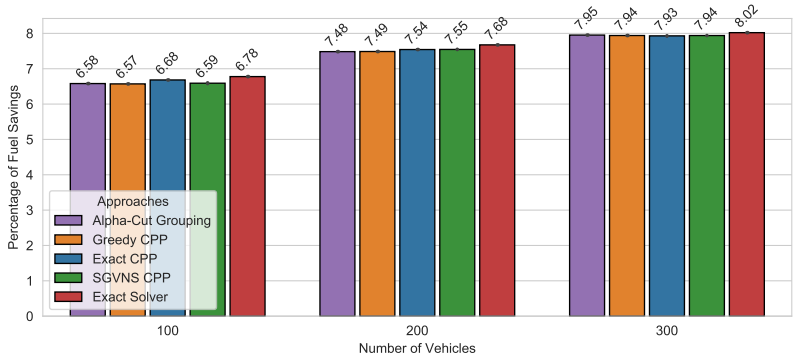


FIGURE 8.22: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Swedish road network.

30×30 Grid Road Network Instance

Small size road networks were used in previous experiments. Now, we want to investigate the performance of the algorithms on **more extensive and complex** road networks. The instances with 10, 20, 30, 50, 100, 200, and 300 vehicles are still used.

Fig. 8.23 and Fig. 8.24 presents the performance on the 30×30 grid road network with 900 nodes and 3480 edges. The execution time for ten vehicles shows no significant differences, and the *SGVNS CPP* and *Exact CPP* algorithms achieve the optimal solution, and the *Greedy CPP* algorithm achieves the near-optimal solution with $\theta^{greedy} = 2.04\%$. The solution which was found by *α-cut Method* is significantly worse than the solutions of the other methods. The instance with 20 vehicles also shows no significant differences in terms of the execution time. The obtained solutions by the *α-cut Method* and *Greedy CPP* are fundamentally worse when compared with the optimal solution. For the instance with 30, vehicles the *Exact Solver* takes the

longest time to determine the solution with 210.9 seconds on average. Slightly faster are *Exact CPP* and *SGVNS CPP* algorithms, which also find the optimal solution. The *Greedy CPP* algorithm is three times faster than the *Exact Solver*. For the instance with 50 vehicles, the previous trend becomes even more evident. The *Exact Solver* takes 849 seconds on average to find an optimal solution. *Exact CPP* and *SGVNS CPP* find the near-optimal solution with $\approx$ 200 seconds faster than the *Exact Solver*. *Greedy CPP* algorithm finds the heuristic solution with 5.94% savings within 279.53 seconds and the α-*cut Method* determines a solution with 4.84% savings within 42.12 seconds.

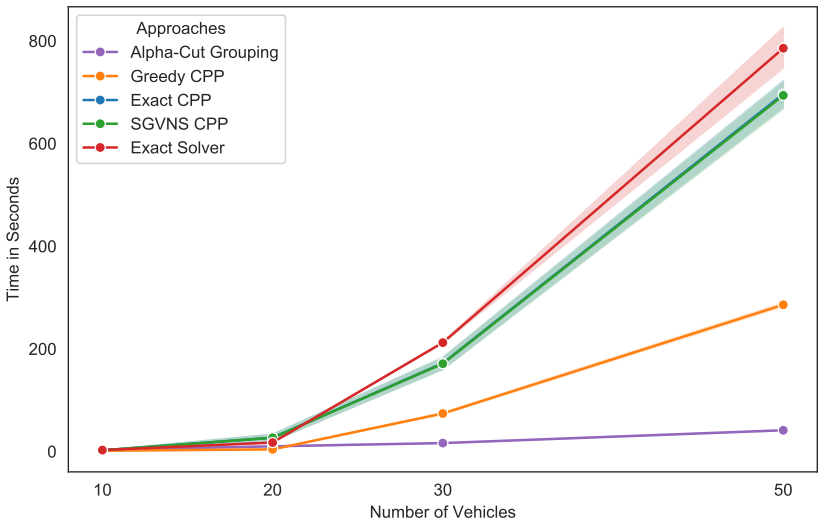


FIGURE 8.23: Execution times of the grouping-based routing algorithms: α-*cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 10, 20, 30, and 50 vehicles on the 30 $\times$ 30 grid road network.

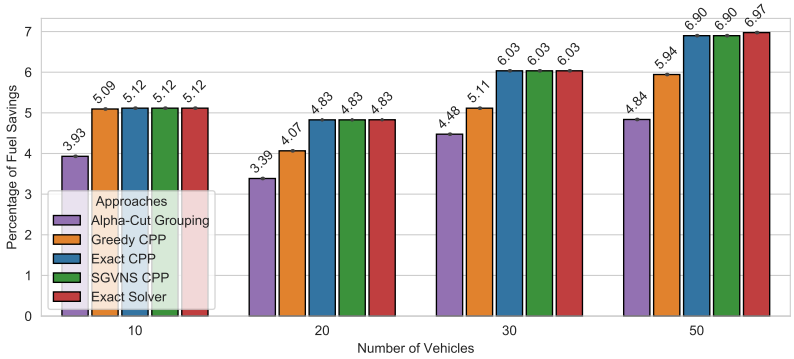


FIGURE 8.24: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 30 $\times$ 30 grid road network.

Fig. 8.25 and Fig. 8.26 presents performance of the *Exact Solver* and our grouping algorithms, resulting in the apparent necessity of heuristics. The *Exact Solver* needs 3536.20 seconds on average to optimally solve the instance with 100 vehicles. *SGVNS CPP* algorithm needs only half of the time to solve the same instance and

achieves 0.31% fewer savings. For the same instance, the *Exact CPP* and *Greedy CPP* take 1950.68 and 681.38 seconds and achieve 7.62% and 7.17% savings. The computation time of the *Exact Solver* increases tremendously for the instances with a larger number of vehicles. The *SGVNS CPP* algorithm, compared to the *Exact Solver*, needs 267% more computational time for the instance with 200 vehicles. In this context, the *Exact Solver* takes 12856.08 seconds to find the corresponding solution with 8.74% savings. The *SGVNS CPP* and *Greedy CPP* take 4814.88 and 4110.24 seconds with 8.41% and 8.28% savings. The total execution time of the route computation with the *Exact CPP* algorithm takes 15812.17 seconds. The most time is needed to form the groups, and in this case, the time limit of 10800 seconds is reached. The rest of the time is used to calculate platoon routes for the groups. The *α-cut Method* achieves only 7.48% savings within 286 seconds. The run time complexity of the *Exact Solver* explodes with the number of vehicles $|H|$. For the instance with 300 vehicles, the *Exact Solver* needs 38494.17 seconds. *SGVNS CPP* and *Greedy CPP* find high-quality solutions up to three times faster than the *Exact Solver*. The total execution time of *Exact CPP* for the instance with 300 vehicles is reduced. The grouping phase reaches the time limit and produces small groups, and the platooning routing is calculated within 20.79 seconds. In total, the *α-cut Method* takes 600 seconds on average and also reaches the optimal solution. The time complexity increases with the number of vehicles, the number of nodes or edges, and **the complexity of the road network structure**. The 30×30 grid road network has fewer nodes and edges compared to the road network of Sweden. The 30×30 grid road network structure is more complex with a larger number of crossings and junctions.

TABLE 8.7: Execution times of grouping-based routing algorithms in comparison to baseline on the 30×30 Grid Road Network.

	α-cut Method		*Greedy CPP*		*Exact CPP*		*SGVNS CPP*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	3.46538	21.0571	1.57667	-44.9218	2.34368	-18.1273	2.35304	-17.8006	2.8626
20	10.1541	-42.1178	4.1904	-76.1131	26.9721	53.7514	26.8142	52.8513	17.5427
30	16.4608	-92.2436	74.079	-65.0937	171.944	-18.9795	171.172	-19.3428	212.222
50	41.4017	-94.7312	285.918	-63.6137	696.557	-11.3552	693.897	-11.6938	785.785
100	83.7601	-97.6169	681.388	-80.6136	1950.69	-44.5002	1594.4	-54.637	3514.76
200	288.28	-98.1767	4262.01	-73.0435	15669.6	-0.892775	4699.79	-70.2747	15810.7
300	599.875	-97.5715	17148.1	-30.5797	11845.1	-52.0478	12777.4	-48.2736	24701.8

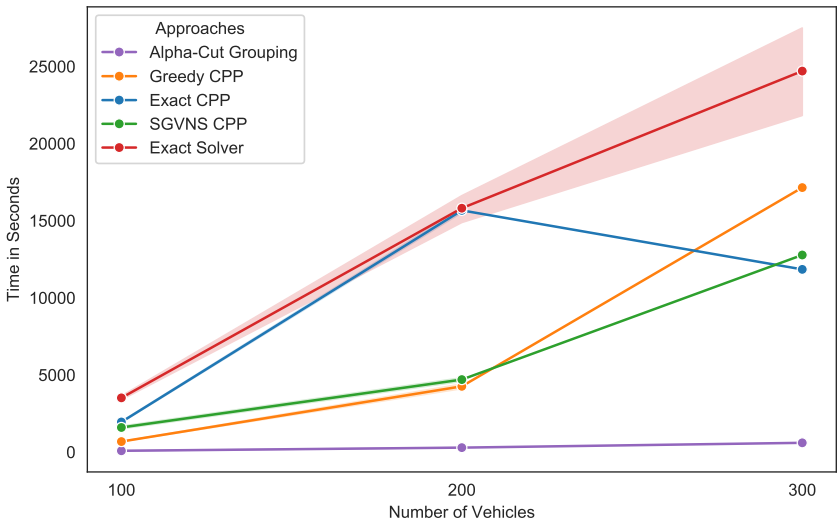


FIGURE 8.25: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 100, 200, and 300 vehicles on the 30×30 Grid road network.

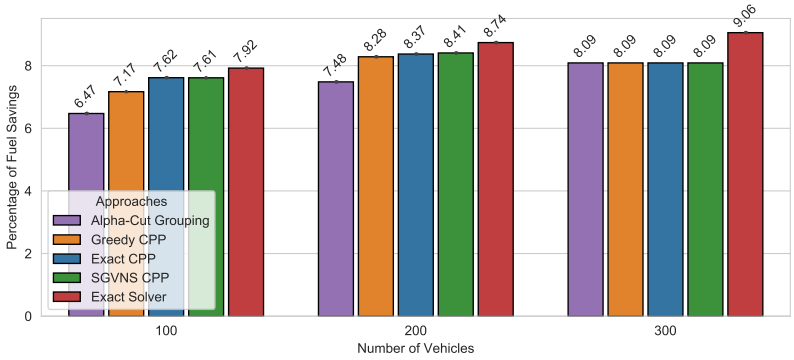


FIGURE 8.26: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 30×30 Grid road network.

German Road Network Instance

Fig. 8.27 and Fig. 8.28 shows the achieved savings and total execution times of the *Exact Solver* and grouping algorithms of the Germany's road network. The report consists of instances with 10, 20, 30, and 50 vehicles. The *Exact Solver* performs significantly slower than the grouping algorithms, statistically verified by the Friedman test with the Bergmann-Hommel post hoc method. Potentially fewer savings can be achieved for Germany's road network due to the more complex topological and geometrical structure. For the instance with 20 vehicles, the *Exact Solver* takes at least ≈ 3.6 times more computational time than the grouping algorithms. Furthermore, grouping algorithms achieve optimal or near-optimal solutions. In the instance with 30 vehicles, the difference grows between heuristic approaches and *Exact Solver* in terms of execution time. The *SGVNS CPP* and *Exact CPP* approaches demonstrate good results when 30 vehicles are divided into three or four groups. This creates

enormous performance with the difference being about ≈ 400 seconds. For the instance with 50 vehicles, the difference between *Exact Solver* and heuristic approaches keeps growing.

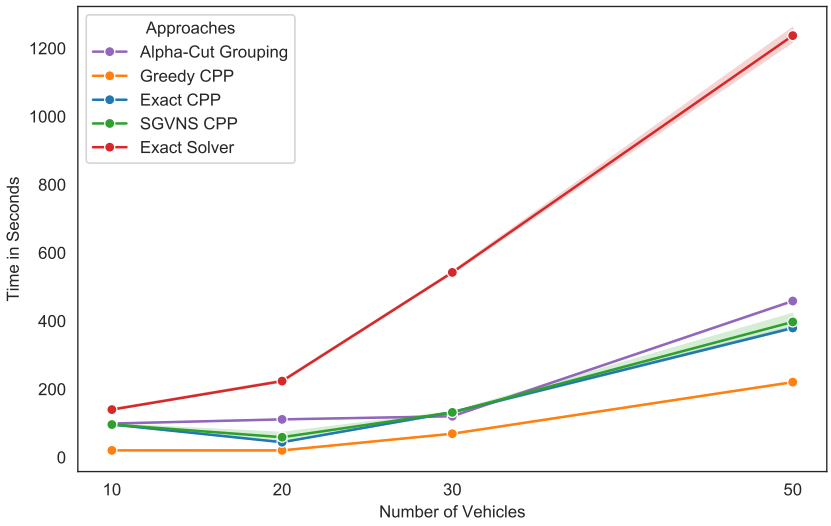


FIGURE 8.27: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 10, 20, 30, and 50 vehicles on the German road network.

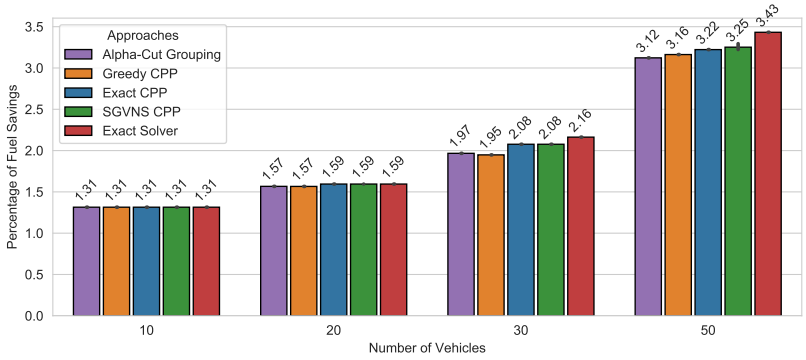


FIGURE 8.28: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the German road network.

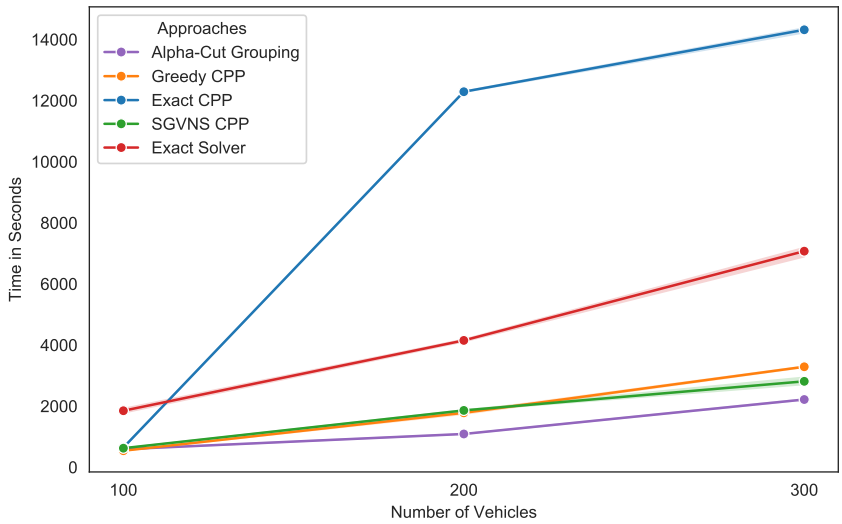


FIGURE 8.29: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 100, 200, and 300 vehicles on the German road network.

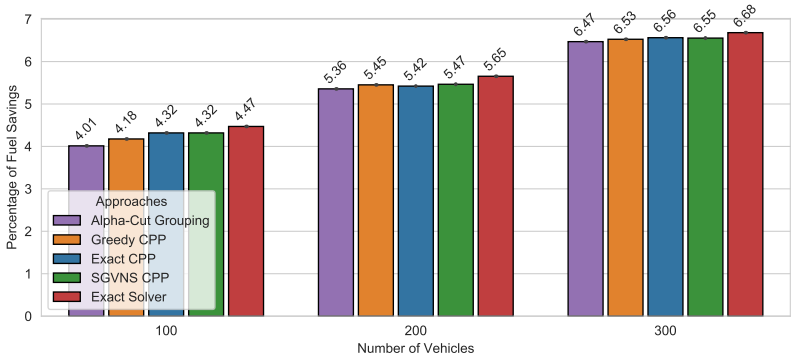


FIGURE 8.30: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the German road network.

For the larger instances with up to 100 vehicles, the computational differences become more apparent in the results. The *Exact Solver* takes 1914.66 seconds on average for the instance with 100 vehicles. The *α-cut Method*, *Greedy CPP*, *Exact CPP*,

TABLE 8.8: Execution times of grouping-based routing algorithms in comparison to the baseline of the German Road Network.

	α-cut Method		*Greedy CPP*		*Exact CPP*		*SGVNS CPP*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	99.0845	-29.2592	20.4058	-85.4314	96.4972	-31.1063	96.1682	-31.3413	140.067
20	111.367	-50.2106	20.3341	-90.9091	44.3373	-80.1779	59.0522	-73.5993	223.676
30	120.485	-77.789	69.312	-87.2225	132.323	-75.6066	131.893	-75.6859	542.454
50	458.535	-62.9364	220.591	-82.1695	379.464	-69.3278	397.251	-67.89	1237.16
100	589.947	-68.0941	539.93	-70.7992	644.931	-65.1204	623.007	-66.3061	1849.02
200	1088.43	-73.7794	1780.33	-57.1116	12290.8	196.088	1859.46	-55.2053	4151.06
300	2216.44	-68.6591	3287.85	-53.5092	14318	102.459	2811.54	-60.2443	7072.04

and *SGVNS CPP* algorithms need $\approx 69\%$, $\approx 71\%$, $\approx 66\%$, and $\approx 68\%$ seconds less time. *Exact CPP* and *SGVNS CPP* achieve 4.32% savings and the other algorithms even less with $\theta^{greedy} = 4.18\%$ and $\theta^{\alpha-cut} = 4.01\%$. The performance of the heuristic solution can be improved by changing the parameters. The execution time for the instance with 200 vehicles increases to ≈ 4151 seconds on average. *SGVNS CPP*, *Greedy CPP*, and α*-cut Method* achieve 5.47%, 5.42%, and 5.45% savings within 1859, 1780, and 1088 seconds. The execution time of the *Exact Solver* with 7072 seconds increases faster than the execution time of *SGVNS CPP*, *Greedy CPP*, and α*-cut Method* algorithms with 2811, 3287, and 2216 seconds for the instance with 300 vehicles.

Southern US Road Network Instance

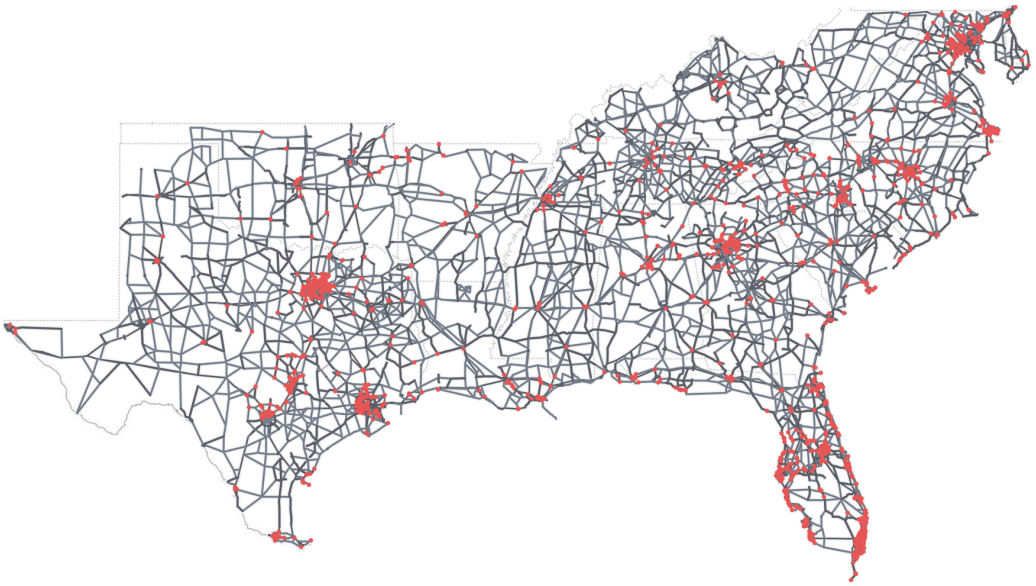


FIGURE 8.31: The road network of the Southern US with 133174 nodes and 280551 edges is generated from OSM with [90].

The Southern US road network is the largest road network with 133174 nodes and 280551 edges in our experiments. We want to show that our heuristic algorithms can "easily" scale up to large road networks. Most existing methods that solve the VRP or VRP related problems cannot support vast road networks. In Fig. 8.32 is it immediately apparent that the *Exact Solver* scales very poorly. For the instance with ten vehicles, the *Exact Solver* needs **significantly more** execution time with 669 seconds on average than the grouping algorithms. *SGVNS CPP*, *Exact CPP*, *Greedy CPP*, and α*-cut Method* achieve optimal solutions with 1.15%. The probability is low that few vehicles can platoon together on a large road network. Therefore the savings increase with the number of vehicles. 1.61% savings are achieved for 20 vehicles instance by the *Exact Solver* within 1435 seconds. The *Exact CPP* and *SGVNS CPP* algorithms reach the optimal solution within 200 and 222 seconds. The *Greedy CPP* and α*-cut Method* obtain 1.52% and 1.48% savings within 51 and 120 seconds. The execution time of the *Exact Solver* increases enormously compared to the grouping algorithms. The *Exact Solver* needs 89% more time than the *SGVNS CPP* to optimally solve the instance with 30 vehicles. The other grouping algorithms need even less time and achieve optimal or near-optimal solutions. For the instance with 50 vehicles, the trend is clearly confirmed because the execution time of the *Exact Solver* has

increased by a factor of 8.4. The *SGVNS CPP* compared to the *Exact Solver* is $\approx 88\%$ faster. The heuristic approaches *SGVNS CPP*, *Exact CPP*, and *Greedy CPP* deliver optimal solutions with 2.38% savings. The α-*cut Method* achieves fewer savings with 2.33% savings. The execution time of the *Exact Solver* is **significantly** worse than that of the heuristics.

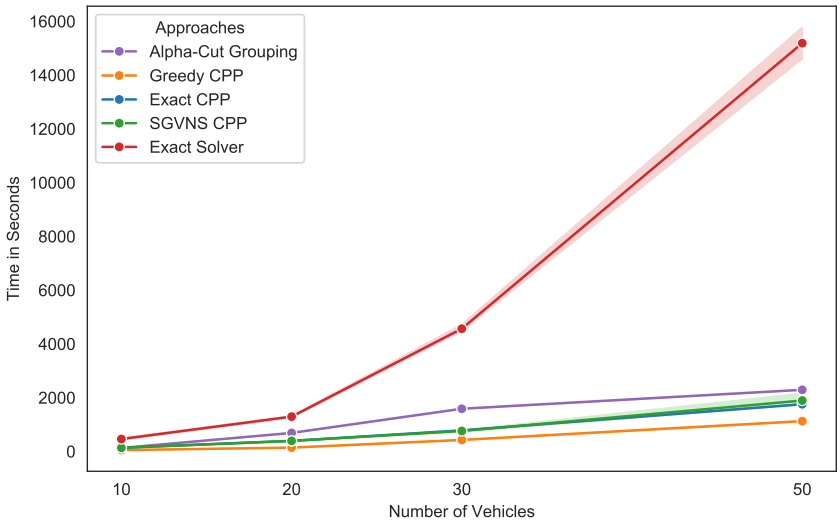


FIGURE 8.32: Execution times of the grouping-based routing algorithms: α-*cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 10, 20, 30, and 50 vehicles on the Southern US road network.

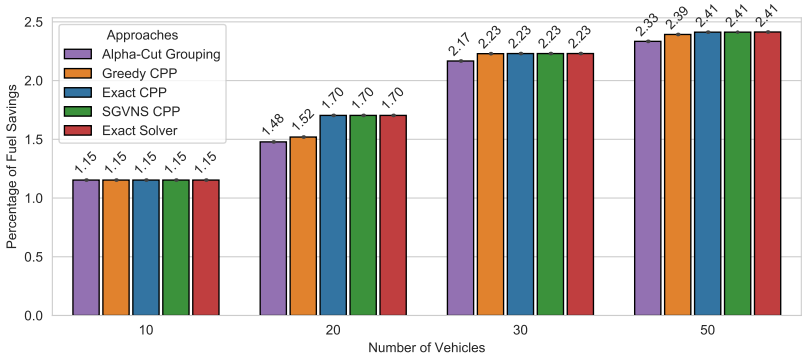


FIGURE 8.33: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Southern US road network.

Fig. 8.34 and Fig. 8.35 shows the performance of grouping algorithms on the Southern US road network for the greater vehicle instances.

The VPP's computational complexity and interactability, along with the necessity of the proposed heuristics, become apparent in these diagrams. The *Exact Solver* is unable to solve instances with 100, 200, and 300 vehicle, because the **available memory** is exhausted. All necessary parameters at Gurobi Optimizer have been adjusted to reduce memory usage (max. 64 GB), but there is not enough memory. Therefore, the *Exact Solver* cannot determine any feasible solution, and heuristic solutions can

only be compared. Solving the instance with 100 vehicles takes 2349, 3465, 3573, and 4293 seconds with the *Greedy CPP*, *Exact CPP*, *SGVNS CPP*, and α*-cut Method* algorithms. The grouping algorithms provide promising results, with up to 3.49% savings. Our vehicle grouping approaches can solve even larger instances. The produced solutions by *Exact CPP* and *SGVNS CPP* of the instance with 200 deliver on average 4.26% and 4.23% savings within 13363 and 11589 seconds. The *Greedy CPP* and α*-cut Method* achieve slightly fewer savings within 11965 and 9279 seconds. The execution times resulted from the solving instance with 200 vehicles have a wide range. *Exact CPP* reaches the time limit after 10800 seconds, and the route calculation takes additional ≈ 50864 seconds. The total solving time of *SGVNS CPP* is 46272 seconds, with 5.21% savings. The *Greedy CPP* takes substantially less time to solve the instance with 300 vehicles and achieves 5.27% savings. A **penalty factor** can adjust the group's forms and sizes, incentive methods and/or incentive parameters. We will discuss the meaning of penalty the factor impact in more detail later on.

TABLE 8.9: Execution times of grouping-based routing algorithms in comparison to the baseline of the Southern US Road Network.

	α*-cut Method*		*Greedy CPP*		*Exact CPP*		*SGVNS CPP*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	128.037	-72.2438	49.1338	-89.3486	134.833	-70.7704	135.197	-70.6916	461.29
20	688.805	-46.828	142.244	-89.0195	392.523	-69.6993	392.431	-69.7064	1295.43
30	1588.12	-65.2264	427.617	-90.6369	783.793	-82.838	760.218	-83.3542	4567.03
50	2292.58	-84.9058	1126.94	-92.5803	1755.71	-88.4405	1893.66	-87.5322	15188.4
100	4276.33	NaN	2374.99	NaN	3478.99	NaN	3454.39	NaN	NaN
200	9123.15	NaN	11729.2	NaN	12677.4	NaN	11291.1	NaN	NaN
300	11759.4	NaN	32361.4	NaN	60634.2	NaN	46858.7	NaN	NaN

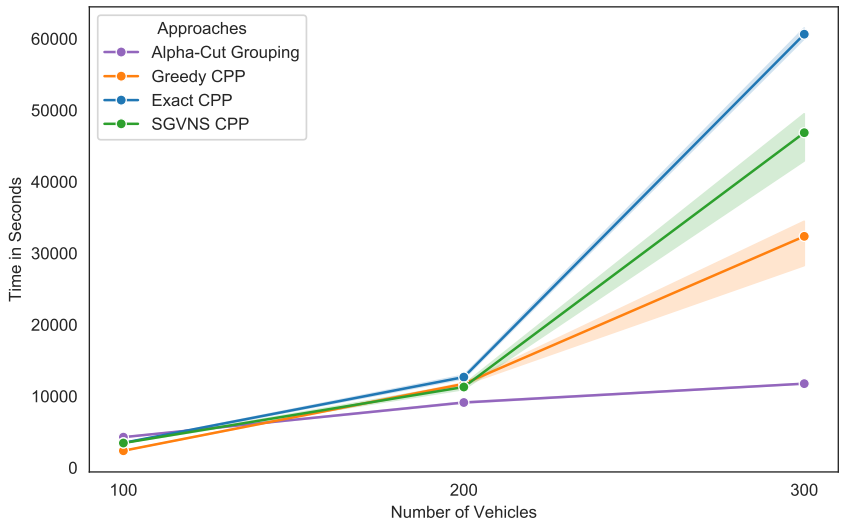


FIGURE 8.34: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* for the instances with 100, 200, and 300 vehicles on the Southern US road network.

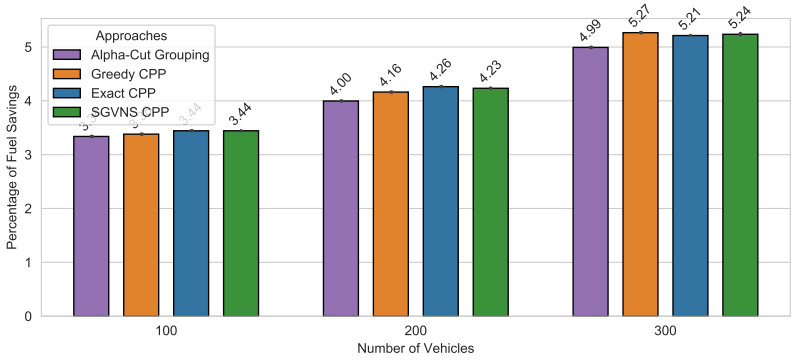


FIGURE 8.35: Fuel savings achieved by the grouping-based routing algorithms and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Southern US road network.

8.2.1 Summarizing Experimental Results of Grouping Algorithms

Section 8.2 reports the experimental results for grouping algorithms *α-cut Method*, *Greedy CPP*, *Exact CPP*, and *SGVNS CPP* compared with *Exact Solver*. We show the excellent performance of our heuristic algorithms with great ability to scale for many vehicles and a large number of road networks. In most cases, grouping-based routing algorithms achieve optimal or near-optimal solutions. For the 20×20 grid, 30×30 grid, German and Southern US road networks, we observe a **significant** statistical difference of solution quality. The solutions of the *Greedy CPP* and *α-cut Method* have particularly strong deviations. A possible reason for this is the poor quality of the initial solutions; in other words, the shortest paths of generated vehicles have small overlap. *Exact CPP* and *SGVNS CPP* approaches are better at dealing with it and achieve, in most cases, near-optimal solutions. Furthermore, there is no significant difference between *Exact CPP* and *SGVNS CPP*. Obtained savings increase with the

number of vehicles $|H|$, and Eq. 8.2 describes the limiting behavior of potential savings. With the number of vehicles, the probability that the platoons can be formed spontaneously also increases. On small road networks like Berlin, this effect is particularly noticeable. Spontaneous platooning on road networks with grid topology is less likely because many similar paths exist in such road networks. For this reason, the execution time of the *Exact Solver* has increased so dramatically in Fig. 8.25.

$$\lim_{|H|\to\infty} f(|H|) = \sum_{h\in H} c(P_h) \cdot \frac{1}{1-\eta} \tag{8.2}$$

Table 8.10 illustrates execution times of incentive calculations for the instances with 10, 20, 30, 50, 100, 200, and 300 vehicles. The data is aggregated from the previous experiments with different road networks and grouped by the vehicle set size. A low standard deviation indicates that the time complexity of incentive calculation is independent of the size of the road network. This property is essential for scalability because the complexity of most other algorithms depends on the size of the road network. The execution time increases rapidly with the number of vehicles. For example, for 300 vehicles, 44850 calculations must be performed. For the instances with a large number of vehicles on small road networks, our grouping-based routing algorithms need more time than the *Exact Solver*. In this aspect, there is excellent potential for optimization. Incentive calculations are weakly correlated; therefore, the calculations can be performed in parallel. For each vehicle pair, we compute three different incentives, as seen in Section 5.1.1, Section 5.1.2, and Section 5.1.2. In some cases, a particular incentive calculation can exclude the other incentive calculation.

TABLE 8.10: Aggregated execution times from the previous experiments of incentive algorithms.

$\|H\|$	Mean	Median	σ	$10th$ Percentile	$90th$ Percentile
10	0.588312	0.578748	0.128976	0.487288	0.630656
20	2.313031	2.377231	0.245179	2.008480	2.458636
30	5.131239	5.257746	0.445305	4.495224	5.540213
50	14.327562	14.690203	1.101879	12.739247	15.340332
100	58.432322	59.818130	4.290779	53.103516	63.287842
200	233.555347	238.773841	13.975089	211.520946	248.785314
300	523.785637	536.136880	31.372626	472.465828	554.194889

Table 8.11 shows aggregated execution times of grouping algorithms with mean, median, standard deviation, $10th$, and $90th$ percentile. The finding of "functional" groups for platooning purposes is not a trivial task. Previously proposed and evaluated grouping algorithms showed varying performance, exceptionally the *Exact CPP* was noticeable. The execution times for the instances with $10, 20, 30$, and 50 vehicles show no essential differences so that the grouping-based routing algorithms can perform efficiently. For the larger vehicle instances, the *Exact CPP* algorithm shows poor performance, and the critical times are marked in red. The average of the measured execution times grows extremely with the number of vehicles. For the instances with 100 and 200 vehicles, we can observe large standard deviations, which reflect that the time complexity of *Exact CPP* also depends on the incentive values and accordingly on the road network instance. For the instance with 300 vehicles, the algorithm reaches the time limit in most cases. This makes it challenging

to apply the algorithm for larger vehicle instances *SGVNS CPP* is an excellent alternative to the *Exact CPP* algorithm, and there are no significant statistical differences in terms of quality with a significance level of $\alpha = 0.01$.

TABLE 8.11: Aggregated execution times from the previous experiments of grouping algorithms.

$\|H\|$	Grouping Algorithms	Mean	Median	σ	$10th$ Percentile	$90th$ Percentile
10	*α-cut Method*	0.051251	0.038557	0.032933	0.034901	0.069231
	Greedy CPP	0.010667	0.008196	0.005963	0.005594	0.018062
	Exact CPP	0.027420	0.019553	0.032698	0.017703	0.024049
	SGVNS CPP	0.009952	0.008123	0.005631	0.006855	0.018789
20	*α-cut Method*	0.102544	0.100535	0.018586	0.080123	0.126449
	Greedy CPP	0.019568	0.018735	0.005883	0.013229	0.028134
	Exact CPP	0.122753	0.121060	0.010829	0.112874	0.131373
	SGVNS CPP	0.034659	0.037586	0.014892	0.017485	0.051515
30	*α-cut Method*	0.152780	0.152455	0.018195	0.131706	0.179098
	Greedy CPP	0.030839	0.030024	0.005821	0.025365	0.037418
	Exact CPP	0.408466	0.404993	0.014893	0.392973	0.425428
	SGVNS CPP	0.063457	0.077454	0.023982	0.031470	0.085076
50	*α-cut Method*	0.303944	0.325670	0.054966	0.222608	0.357543
	Greedy CPP	0.062104	0.060531	0.007349	0.054059	0.071708
	Exact CPP	2.295520	2.195472	0.223703	2.149762	2.783401
	SGVNS CPP	0.207002	0.227026	0.052476	0.089248	0.241475
100	*α-cut Method*	0.996281	0.969917	0.128992	0.845622	1.187177
	Greedy CPP	0.216317	0.209478	0.027639	0.187729	0.261922
	Exact CPP	526.931150	70.843619	878.308583	19.043606	2599.988516
	SGVNS CPP	1.174075	1.157880	0.066562	1.115540	1.283283
200	*α-cut Method*	2.938091	2.856636	0.998300	1.420585	4.167298
	Greedy CPP	1.018600	0.913550	0.254823	0.790730	1.371304
	Exact CPP	5491.567849	2539.645161	4916.774755	230.031298	10900.294755
	SGVNS CPP	7.981112	7.587610	1.432381	6.741149	9.666350
300	*α-cut Method*	6.598154	7.172895	1.487687	4.371214	7.840812
	Greedy CPP	3.171901	2.802709	1.550521	1.860487	5.364766
	Exact CPP	11185.487471	11144.669169	71.127513	11126.802997	11291.437851
	SGVNS CPP	24.607794	23.354176	3.422670	21.054105	28.718393

Statistical Analysis of Grouping-based Routing Algorithms in Terms of Execution Times

We have divided the problem instances into three difficulty classes depending on the size of the road networks. The grouping-based routing algorithms provide different performances in different difficulty classes. The 10×10 grid, 20×20 grid, and Berlin road networks are the most uncomplicated instances. The *Exact Solver* showed the best performance for the problems in this difficulty class, and our grouping-based routing algorithms showed the worst performance due to expensive incentive calculations. This statement can be confirmed by the statistical test, which should show that the proposed approaches differ in terms of execution time. The null hypothesis H_0 represents the absence of a difference between approaches in terms of execution time. We use the *Friedman's* test with *Bergmann-Hommel* post hoc procedure for the multiple comparisons [31, 45, 21]. Table 8.12 shows the adjusted *p*-values in the form of a matrix determined by *Friedman's* test. The H_0 is rejected for all algorithm pairs with a level of significance at $\alpha = 0.05$. Now we can conclude that all proposed algorithm have a significant difference in terms of execution time. The *Exact Solver* solves the problem instances from the most straightforward difficulty class more efficiently than the proposed grouping-based routing algorithms.

The Bavarian and Swedish road networks are the instances with average difficulty. We applied *Friedman's* test with the *Bergmann-Hommel* post hoc procedure

TABLE 8.12: Adjusted *p*-values of the Friedman test with Bergmann-Hommel post hoc procedure [31, 45, 21]. Test consider test cases for 10, 20, 30, 50, 100, 200, and 300 on 10×10 grid, 20×20 grid, and Berlin road networks. The results of the test shown significant statistical differences among the different algorithms with a level of significance at $\alpha = 0.05$.

	α-cut Method	*Greedy CPP*	*Exact CPP*	*SGVNS CPP*	*Exact Solver*
α-cut Method		0.000	0.355	0.000	0.000
Greedy CPP	0.000		0.000	0.000	0.000
Exact CPP	0.355	0.000		0.000	0.000
SGVNS CPP	0.000	0.000	0.000		0.000
Exact Solver	0.000	0.000	0.000	0.000	

again. The null hypothesis H_0 states that there is no difference between the algorithms in terms of execution time. The results of the test are presented in Table 8.13. The most *p*-values in the Table 8.13 are 0 or < 0.05. The adjusted *p*-value for the *Exact Solver* and *SGVNS CPP* pair is ≥ 0.05, and H_0 can not be rejected. For the instances from the average difficulty class, the *Exact Solver* and *SGVNS CPP* algorithms work similarly in terms of execution time. For such instances, it may be worthwhile to use the grouping-based routing algorithms, e.g., for the road network of Sweden with $10, 20, 30$, and 50 vehicles.

TABLE 8.13: Adjusted *p*-values of the Friedman test with Bergmann-Hommel post hoc procedure [31, 45, 21]. This test considers test cases for 10, 20, 30, 50, 100, 200, and 300 on the Bavarian and Swedish road networks. The result of the test shows significant statistical differences among the grouping-based routing algorithms with a level of significance at $\alpha = 0.05$. The statistical comparison of the *Exact Solver* and *SGVNS CPP* show no significant difference.

	α-cut Method	*Greedy CPP*	*Exact CPP*	*SGVNS CPP*	*Exact Solver*
α-cut Method		0.000	0.030	0.000	0.000
Greedy CPP	0.000		0.000	0.000	0.000
Exact CPP	0.030	0.000		0.000	0.000
SGVNS CPP	0.000	0.000	0.000		0.938
Exact Solver	0.000	0.000	0.000	0.938	

The 30×30 Grid, German, and Southern US road networks are the toughest problem instances and belong to the hard difficulty class. Discussed approaches in related work evaluated their algorithms on considerably simpler road networks. Without grouping-based routing algorithms, it is very difficult or even impossible to solve such instances. We showed that *Exact Solver* needs many resources to determine the solution. Our grouping-based routing algorithms determine a solution much more efficiently. To confirm this statement statistically, we have performed the *Friedman's* test for the 30×30 Grid, German, and Southern US road networks and with different vehicle set sizes. For the Southern US road network, the *Exact Solver* cannot provide solutions for $100, 200$, and 300 vehicle instances. The null hypothesis H_0 is similarly formulated as before and represents the absence of a difference between grouping-based routing algorithms and the *Exact Solver* in terms of execution time. Table 8.14 shows the resulting adjusted *p*-values; all of them are 0 or < 0.05. The *α-cut Method* and *Greedy CPP* algorithms solve complicated instances with the

best run-time efficiency compared to the other algorithms. However, these algorithms provide high-quality solutions with **significantly less savings**. *Exact Solver* and *Exact CPP* approaches solve larger road networks' problems less efficiently than the other grouping-based routing algorithms. For a large number of vehicles, these algorithms are not applicable. *SGVNS CPP* differs from other algorithms in every aspect, both in execution time and achieved savings. *SGVNS CPP* delivers a solution with a great ratio between execution time and solution quality.

TABLE 8.14: Adjusted p-values of the Friedman test with Bergmann-Hommel post hoc procedure [31, 45, 21]. The test considers test cases for 10, 20, 30, 50, 100, 200, and 300 on the 30×30 Grid, German and Southern US road networks. The result of the test shown significant statistical differences among the grouping algorithms with a level of significance at $\alpha = 0.05$.

	α-cut Method	*Greedy CPP*	*Exact CPP*	*SGVNS CPP*	*Exact Solver*
α-cut Method		0.000	0.000	0.001	0.000
Greedy CPP	0.000		0.000	0.000	0.000
Exact CPP	0.000	0.000		0.003	0.000
SGVNS CPP	0.001	0.000	0.003		0.000
Exact Solver	0.000	0.000	0.000	0.000	

8.3 Computational Experiments of Routing Algorithms

The previous experiments show that the grouping-based routing algorithms can solve complex problem instances more efficiently than the *Exact Solver*. In any case, the platooning routes are determined by solving the ILP problem formulated in Section 2.1.1. The difficulty of the ILP problem increases with the number of vehicles, the size of the road network, and other factors. Solving such large-scale problem instances requires excessive memory and CPU time, respectively. Chapter 6 proposes an efficient algorithm for solving large-scale mixed-integer programs, called Column Generation. Our formulation of CG in Section 6.2 is tailored to the Vehicle Platooning Problem. Additionally, we implemented two algorithms from the related work [57], named *Best Pair* and *Hub Heuristic*. The following experiments compare the *Exact Solver*, CG, *Best Pair*, and *Hub Heuristic* performance. Problem instances with varying difficulty are used for the scalability tests.

10×10 Grid Road Network Instance

Fig. 8.36 and Fig. 8.37 present the performance of proposed platooning routing algorithms on the 10×10 road network. Solving the instance with ten vehicles by *Best Pair* algorithm takes 73.53 seconds. The other approaches solve the same instance in less than one **second**. For the instance with 20 vehicles, the *Best Pair* algorithm shows poor efficiency and scalability with 329 seconds execution time. The distance of the *Best Pair* to the optimal solution is 2.48% in savings. The *Hub Heuristic* and CG approaches achieve slightly better results with 3.34% and 3.77% savings. The execution time of *Best Pair* has **doubled** for the instance with 30 vehicles, and the provided solution has poor quality with 3.16% savings. *Hub Heuristic* and CG achieve 4.25% and 4.16% within 6.82 and 1.22 seconds. For the instance with 50 vehicles, the increasing trend is confirmed, especially for *Best Pair* algorithm.

TABLE 8.15: Execution times of **routing** algorithms in comparison to baseline on the 10×10 Grid Road Network.

	Best Pair		*Hub Heuristic*		*Column Generation*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	72.1409	38292.4	0.901538	379.786	0.222797	18.5697	0.187904
20	322.797	103841	2.66851	759.265	0.633623	104.027	0.310558
30	797.945	155156	6.52152	1168.89	0.80731	57.0788	0.513953
50	2364.43	209997	22.9658	1940.67	1.98479	76.3623	1.1254
100	10680.6	554548	169.93	8724.53	16.7735	771.051	1.92566
200	10800.6	248621	1340.32	30765.4	69.4935	1500.32	4.34247
300	10801.5	130736	4521.2	54664.4	315.88	3726.19	8.25573

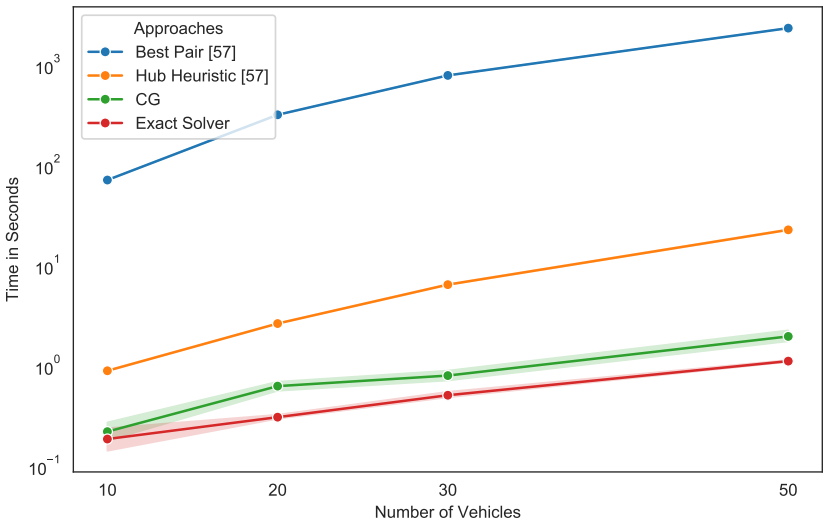


FIGURE 8.36: Execution times of the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 10×10 grid road network.

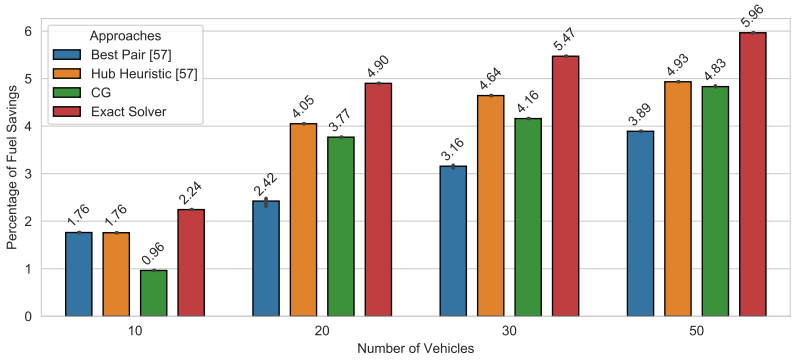


FIGURE 8.37: Fuel savings achieved by the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 10×10 grid road network.

Fig. 8.38 and Fig. 8.39 shows the performance of routing algorithms for the larger vehicle instances on 10×10 grid road network. In the instance with 100 vehicles, the *Best Pair* has a time limit of 10800 seconds and achieves 5.97% savings. The scalability of the *Best Pair* algorithm depends on the number of vehicles. For the instances with 200 and 300 vehicles, the *Best Pair* approach reaches the time limit again without any solutions. The execution time of *Hub Heuristic* also increases rapidly for the instances with 100, 200, and 300 vehicles, and the solving times are 169.3, 1306.4, and 4364.2 seconds. The CG approach shows a rising tendency in terms of execution times with 17.89, 25.73, and 415.08 seconds. *Hub Heuristic* and CG achieve similar results for the instances with 100, 200, and 300 vehicles. The *Exact Solver* finds the optimal solutions under 9 seconds in all cases.

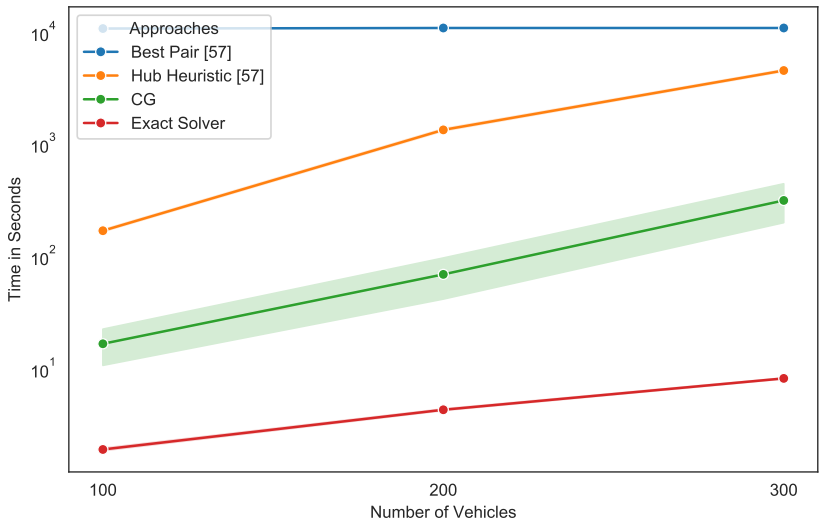


FIGURE 8.38: Execution times of the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 10×10 grid road network.

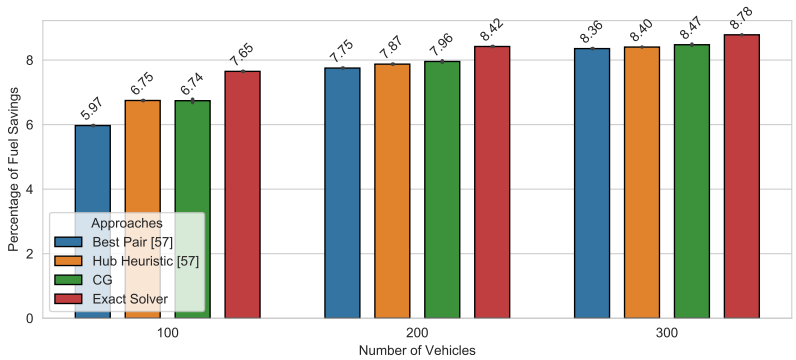


FIGURE 8.39: Fuel savings achieved by the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 10×10 grid road network.

20×20 Grid Road Network Instance

Fig. 8.40 and Fig. 8.41 demonstrate the results on the 20×20 grid road network. Solving the instance with ten vehicles with the *Best Pair* takes 5067 seconds. The algorithm provides a low-quality solution with 0.07% savings. *Hub Heuristic* and CG solve the same instance within 4.8 and 1.2 seconds, thereby 2.05% and 1.83 savings are achieved. For the instance with 20 vehicles, the *Best Pair* reaches the time limit and provides a low-quality result with 0.49% savings. No solutions can be determined for the larger vehicle sets. The execution times of *Hub Heuristic*, CG, and the *Exact Solver* show no significant differences. *Hub Heuristic* and CG achieve acceptable solutions compared with the optimal solutions.

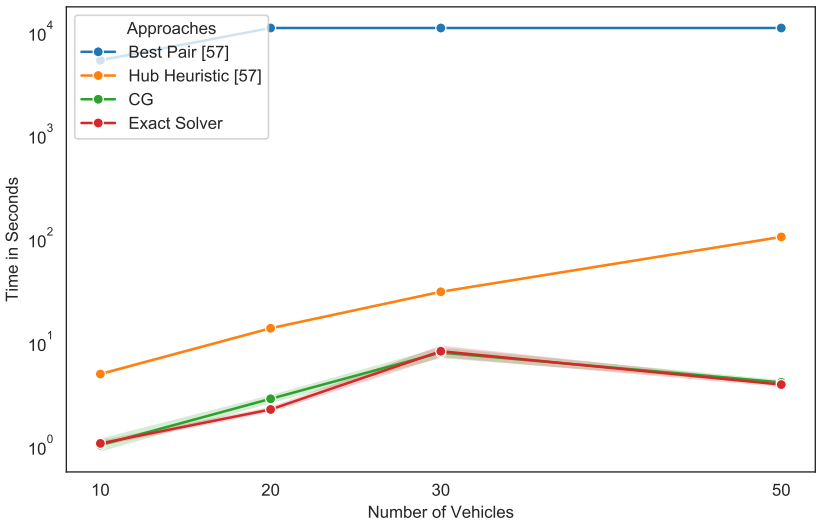


FIGURE 8.40: Execution times of the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 20×20 grid road network.

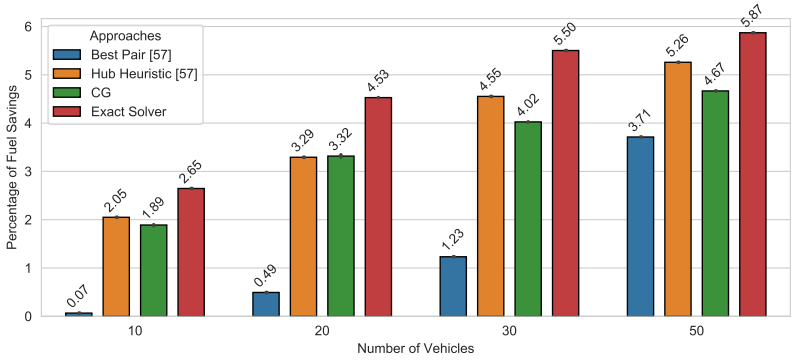


FIGURE 8.41: Fuel savings achieved by the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 20×20 grid road network.

Fig. 8.42 and Fig. 8.43 shows the performance of *Hub Heuristic*, CG, and *Exact Solver*. The *Best Pair* terminates for the 30 and 50 vehicle instances without a solution; therefore, the *Best Pair* is excluded from testing for larger vehicle instances. The execution time of *Hub Heuristic* increases rapidly and shows a significant difference between CG and *Exact Solver*. For the instance with 300 vehicles, the *Hub Heuristic* reaches the time limit, which indicates poor scalability. The achieved solutions by *Hub Heuristic* are slightly better than the CG solutions. The CG demonstrates similar time complexity as the *Exact Solver*.

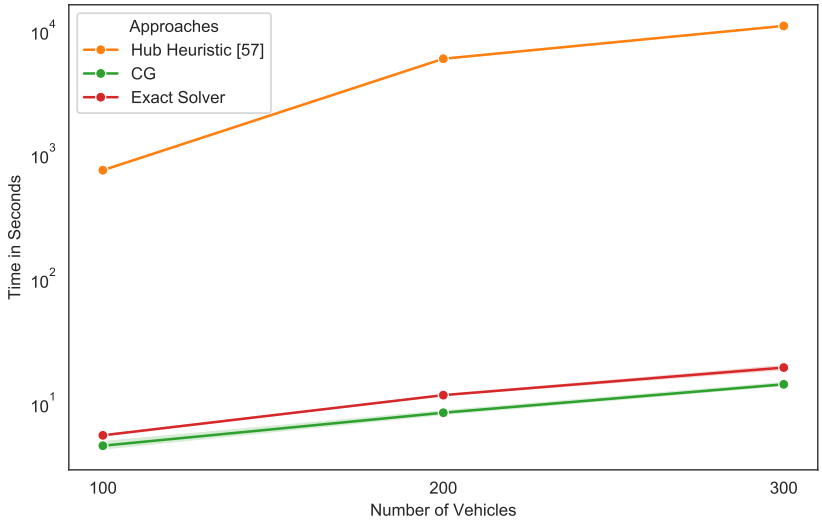


FIGURE 8.42: Execution times of the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 20×20 grid road network.

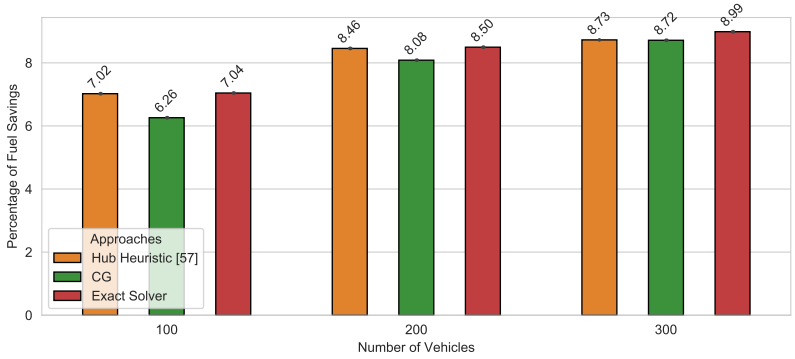


FIGURE 8.43: Fuel savings achieved by the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 20×20 grid road network.

TABLE 8.16: Execution times of **routing** algorithms in comparison to baseline on the 20×20 Grid Road Network.

	Best Pair		*Hub Heuristic*		*Column Generation*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	5275.09	495284	4.97086	366.814	1.02618	-3.63157	1.06485
20	10800.1	477300	13.7317	506.987	2.86762	26.7582	2.26227
30	10800.1	130847	30.8667	274.248	8.00903	-2.8932	8.24765
50	10800.1	274487	104.367	2553.47	4.12898	4.97727	3.93322
100	10800.5	193373	752.091	13372.4	4.60583	-17.4945	5.58245
200	NaN	NaN	5912.12	50269.7	8.4798	-27.7544	11.7375
300	NaN	NaN	10849.9	55396.1	14.3362	-26.6719	19.5508

Berlin Road Network Instance

Fig. 8.44 and Fig. 8.45 provides performance for all routing algorithms on the Berlin road network. The *Best Pair* inefficiently solves the instance with ten vehicles with 2190.83 seconds. For the larger vehicle instances, the *Best Pair* algorithm terminates within a given time limit. The obtained solutions by *Best Pair* algorithm demonstrates high quality. *Hub Heuristic* takes slightly more time to solve the instances with 10, 20, 30, and 50 than the CG and *Exact Solver*.

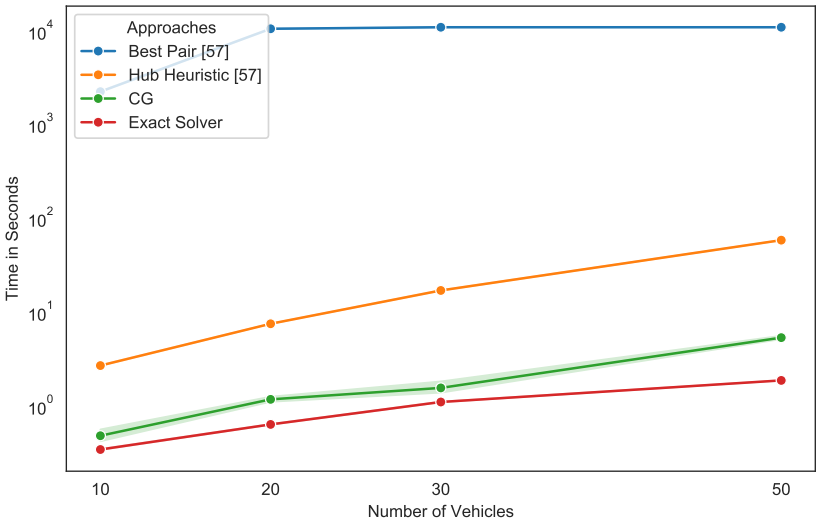


FIGURE 8.44: Fuel savings achieved by the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Berlin road network.

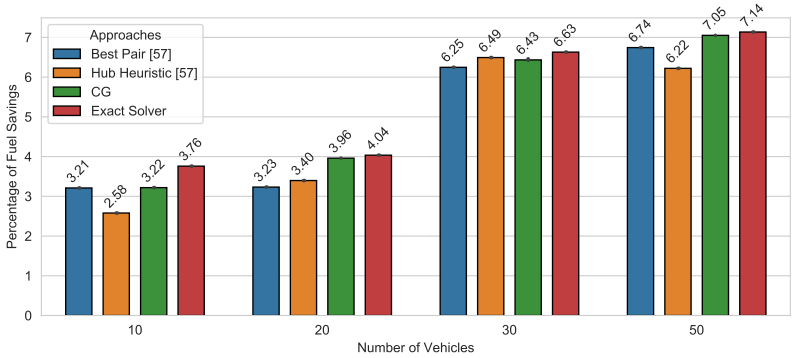


FIGURE 8.45: Fuel savings achieved by the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Berlin road network.

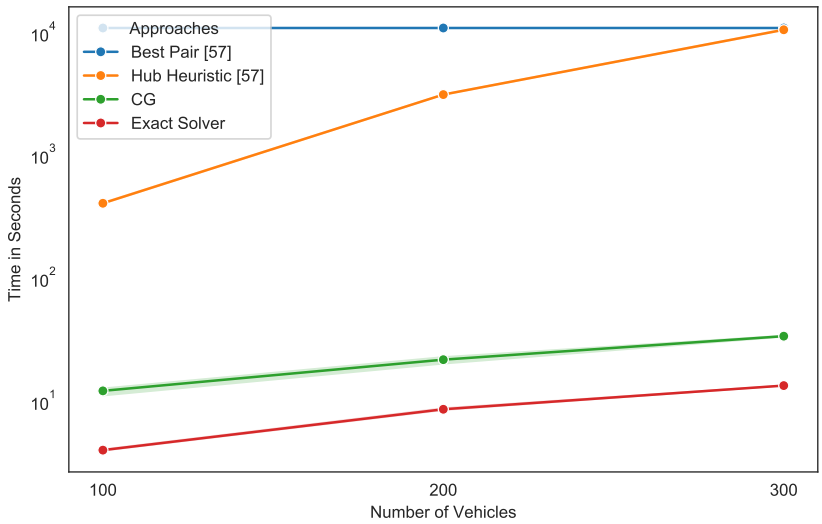


FIGURE 8.46: Execution times of the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Berlin road network.

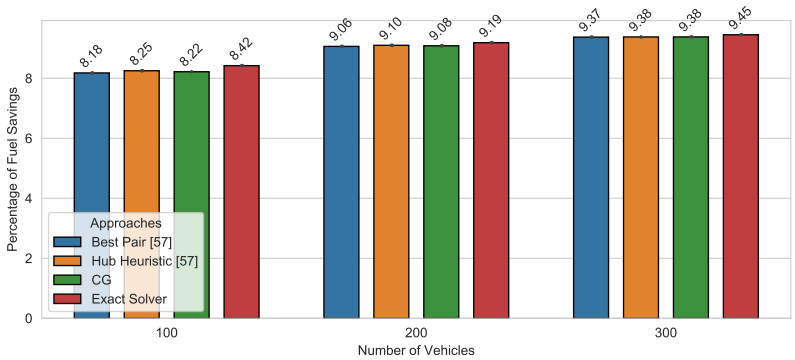


FIGURE 8.47: Fuel savings achieved by the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Berlin road network.

TABLE 8.17: Execution times of **routing** algorithms in comparison to baseline on the Berlin Road Network.

	Best Pair		*Hub Heuristic*		*Column Generation*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	2228.03	644359	2.7058	682.654	0.484646	40.1842	0.345721
20	10404.5	1.62719e+06	7.53479	1078.46	1.18165	84.8124	0.639378
30	10800.1	974854	17.0494	1439.1	1.56443	41.2254	1.10775
50	10800.1	574320	58.4999	3011.41	5.36464	185.327	1.88017
100	10800.2	268710	406.915	10027.8	12.1784	203.11	4.0178
200	10800.7	125043	3108.14	35912.7	21.8257	152.885	8.63068
300	10801.5	80264.8	10444.2	77606	33.8701	151.997	13.4407

Bavarian Road Network Instance

Fig. 8.48 and Fig. 8.49 presents the performance of routing algorithms on the Bavarian road network. The *Best Pair* reaches the time limit for all vehicle sets. *Hub Heuristic* takes 84.67, 183.06, 334.91, and 841.48 seconds to tackle problem instances with 10, 20, 30, and 50 vehicles and provides the worst results. *Exact Solver* solves this instance much more efficiently. The CG approach provides the best execution time and solves the instances optimally.

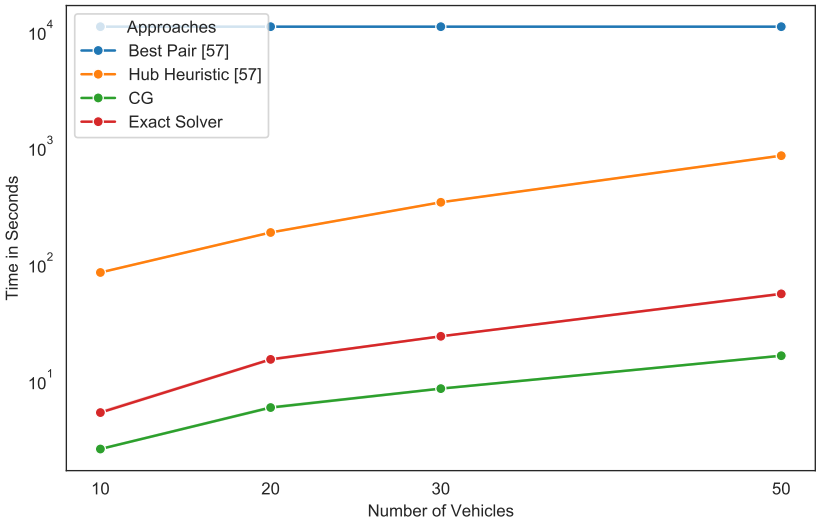


FIGURE 8.48: Execution times of the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Bavarian road network.

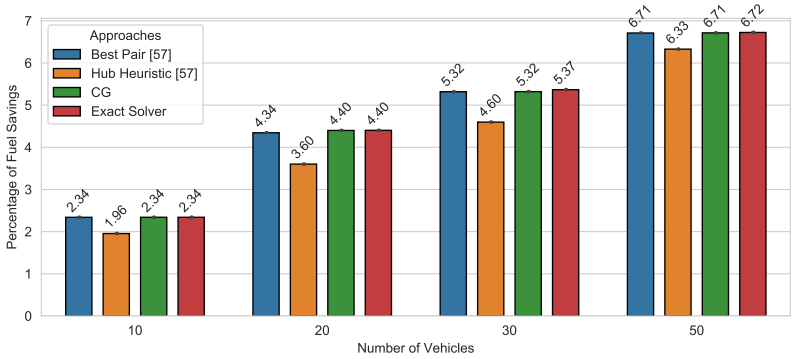


FIGURE 8.49: Fuel savings achieved by the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Bavarian road network.

Fig. 8.50 and Fig. 8.51 illustrates the performance of *Hub Heuristic*, CG, and *Exact Solver* on the Bavarian road network. Due to the poor execution time, we have excluded *Best Pair* from the evaluation. The *Hub Heuristic* shows significant differences in terms of the execution time, solving the instance with 100 vehicles takes 4384.26 seconds. The CG solves the same instance 128× faster and achieves a near-optimal solution with 7.72% savings. *Hub Heuristic* reaches the time limit in the instance with

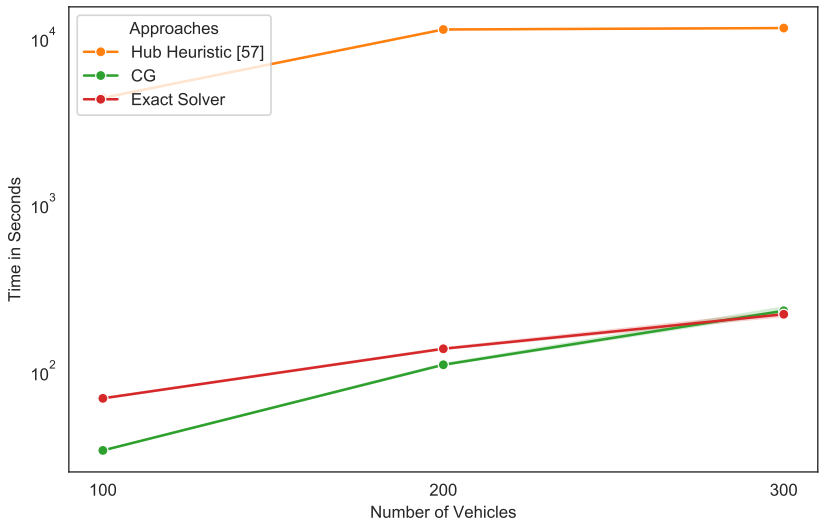


FIGURE 8.50: Execution times of the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Bavarian road network.

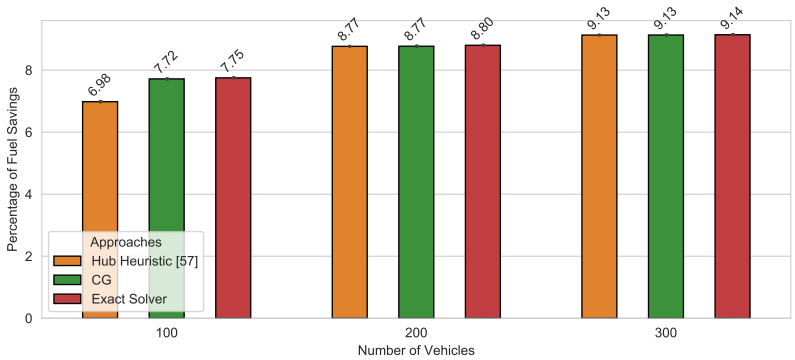


FIGURE 8.51: Fuel savings achieved by the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Bavarian road network.

200 vehicles. CG and *Exact Solver* solve the same instance within 115.72 and 144.88 seconds.

TABLE 8.18: Execution times of **routing** algorithms in comparison to the baseline of the Bavarian Road Network.

	Best Pair		*Hub Heuristic*		*Column Generation*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	10800.2	202080	84.6781	1485.17	2.60234	-51.2843	5.34188
20	10800.3	70727.4	186.339	1121.99	5.89792	-61.322	15.2488
30	10800.2	44811.9	338.114	1306.02	8.56768	-64.372	24.0476
50	10800.3	19344.7	847.949	1426.64	16.3944	-70.4837	55.5435
100	NaN	NaN	4353.5	6169.69	33.8813	-51.2058	69.4372
200	NaN	NaN	11170.4	8026.88	110.265	-19.7781	137.45
300	NaN	NaN	11395.4	5053.98	231.967	4.91528	221.099

Swedish Road Network Instance

Fig. 8.52 and Fig. 8.53 shows the performance of routing algorithms on the Swedish road networks. The *Best Pair* reaches the time limit of 10800 seconds for the instances with 10, 20, 30, and 50, achieved savings are 2.03%, 3.10%, 4.29%, and 4.51%. The execution times of CG and *Exact Solver* increase very slightly. The CG solves instances within 5.4, 10.25, 17.10, and 37.22 seconds and *Exact Solver* solves the same instances within 11.55, 26.89, 40.62, and 95.37 seconds. The *Hub Heuristic* takes significantly more execution time than the CG and *Exact Solver* algorithms. Table 8.19 shows the percentage distribution execution times for the individual steps of *Hub Heuristic*. The initial step consumes most of the time through Dijkstra's algorithm. The percentage share of formation step increases with the number of vehicles. Solving the VPP via hubs takes the least time. All algorithms achieve high-quality solutions.

TABLE 8.19: Percentage distribution of the execution times for the individual steps of *Hub Heuristic*. The instances with 10 - 50 vehicles on Swedish road network is used.

$\|H\|$	Initialization (Lines 3 - 6)	Formation (Lines 7 - 9)	Routing (Lines 10 - 14)
10	$\approx 85\%$	$\approx 3\%$	$\approx 12\%$
20	$\approx 78\%$	$\approx 11\%$	$\approx 11\%$
30	$\approx 71\%$	$\approx 20\%$	$\approx 9\%$
50	$\approx 51\%$	$\approx 43\%$	$\approx 6\%$

Fig. 8.52 and Fig. 8.53 illustrates computational results on the Swedish road network. *Hub Heuristic* takes 10236.38, 12193.04, and 12969.50 seconds for the instances with 100, 200, and 300 vehicles. The execution time of CG grows continuously from 147.60 to 1082.13 seconds. The *Exact Solver* and CG obtain 6.78% and 6.68% savings, while *Hub Heuristic* achieves less savings with 5.95%. The CG and *Hub Heuristic* algorithms almost optimally solve the instances with 200 and 300 vehicles nearly optimal.

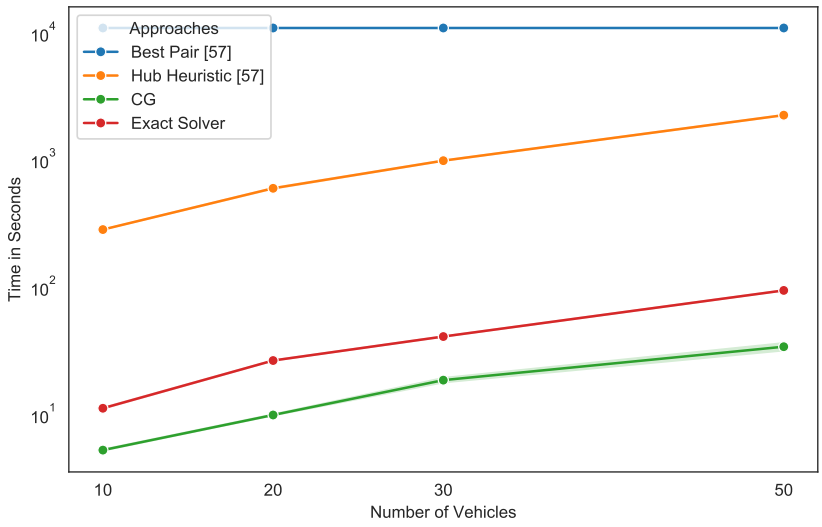


FIGURE 8.52: Execution times of the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Swedish road network.

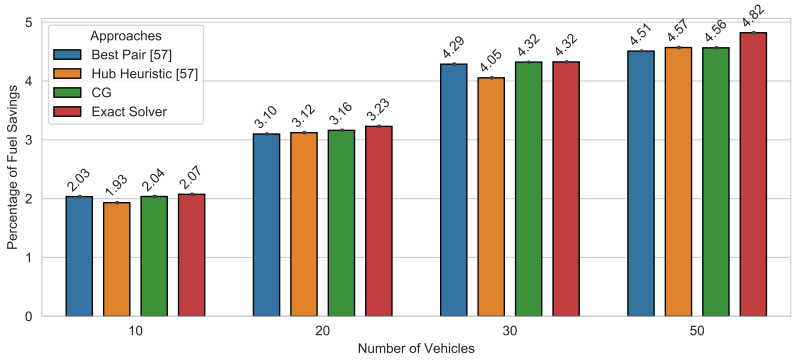


FIGURE 8.53: Fuel savings achieved by the routing algorithms: *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the Swedish road network.

TABLE 8.20: Execution times of **routing** algorithms in comparison to the baseline of the Swedish Road Network.

	Best Pair		*Hub Heuristic*		*Column Generation*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s	overh. %	mean s
10	10800.5	96040.1	283.314	2421.91	5.27502	-53.0446	11.2341
20	10800.4	40494.6	596.982	2143.83	9.94698	-62.613	26.6055
30	10800.5	26222.9	983.342	2296.6	18.6689	-54.5002	41.0307
50	10800.4	11343.1	2237.59	2270.74	34.1445	-63.8237	94.3836
100	NaN	NaN	10194.9	2576.01	190.076	-50.1077	380.974
200	NaN	NaN	12137.1	2657.39	466.518	5.98685	440.166
300	NaN	NaN	12851.3	2866.43	325.046	-24.9703	433.223

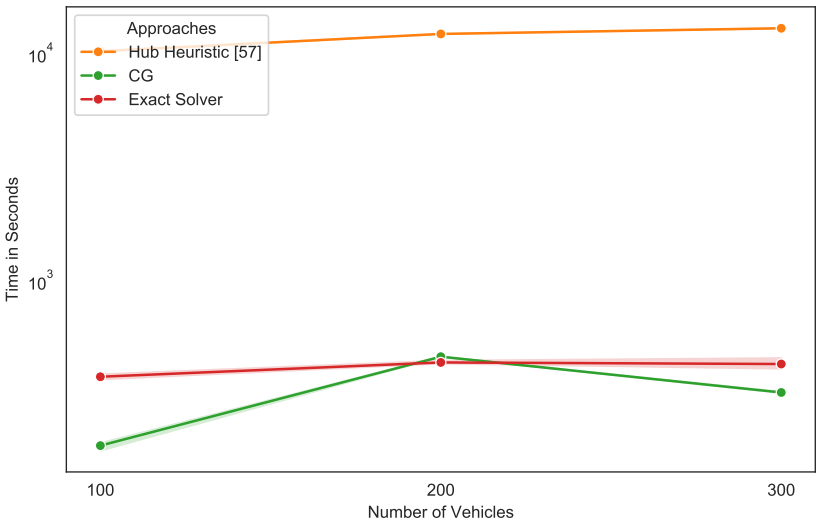


FIGURE 8.54: Execution times of the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Swedish road network.

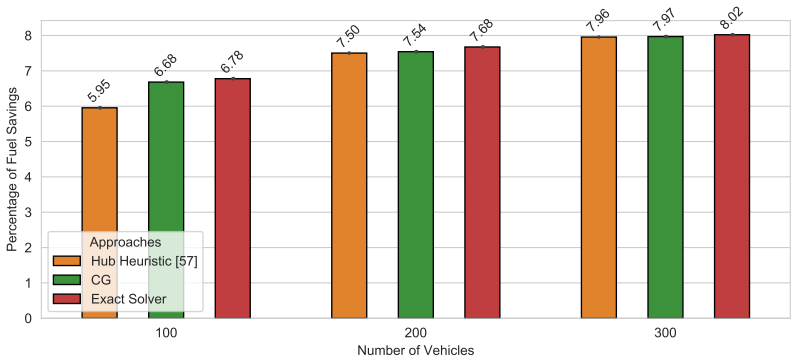


FIGURE 8.55: Fuel savings achieved by the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the Swedish road network.

30×30 Grid Road Network Instance

Fig. 8.56 and Fig. 8.57 presents the performance of the routing algorithms on the 30×30 grid road network. The *Best Pair* algorithm is not evaluated for 30×30 grid, German, and Southern US road networks. The execution time of *Exact Solver* increases rapidly, solving the 50 vehicles instance takes considerably more time than the other algorithms. *Hub Heuristic* algorithm solves the instances with 30 and 50 vehicles more efficiently than the *Exact Solver*. CG approach solves all instances most efficiently, but with an increasing tendency. *Hub Heuristic* reaches near-optimal solution and CG slightly less with 4.35% savings in the instance with ten vehicles. In other cases, CG performs better than the *Hub Heuristic*.

Fig. 8.58 and Fig. 8.59 illustrate experimental results for the routing algorithms performed on the 30×30 grid road network. The execution times of the routing algorithms on the road networks with a grid-like topology increases enormously. The *Hub Heuristic* solves the instance with 100 vehicles most efficiently but achieves

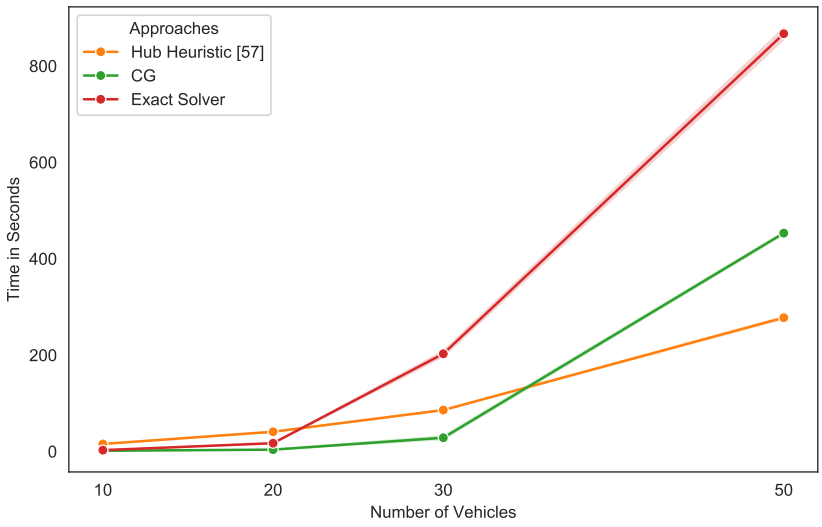


FIGURE 8.56: Execution times of the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 30×30 grid road network.

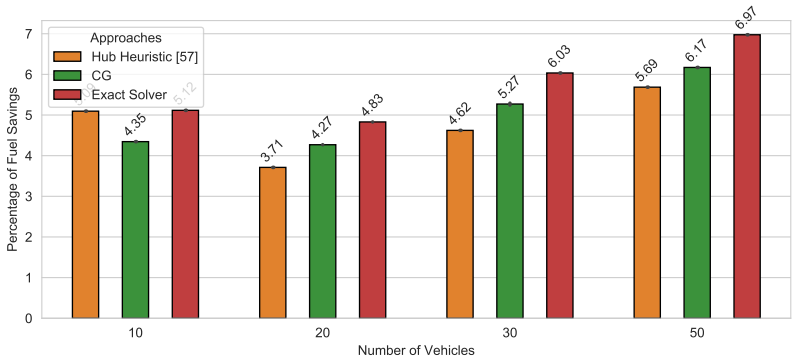


FIGURE 8.57: Fuel savings achieved by the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, and 50 vehicles on the 30×30 grid road network.

the least savings. For the same instance, the CG algorithm takes essentially more time for solving than the *Exact Solver* and *Hub Heuristic* approaches. For the larger instances with 200 and 300 vehicles, the *Hub Heuristic* and CG algorithm reach the time limit. For these instances, CG algorithm provides near-optimal solutions, and *Hub Heuristic* achieves slightly less savings. No time limit is set for the *Exact Solver* to determine the optimal solution. Table 8.21 shows the rapid increase of the execution times from 200 vehicles and that the *Exact Solver* needs more time than the time limit.

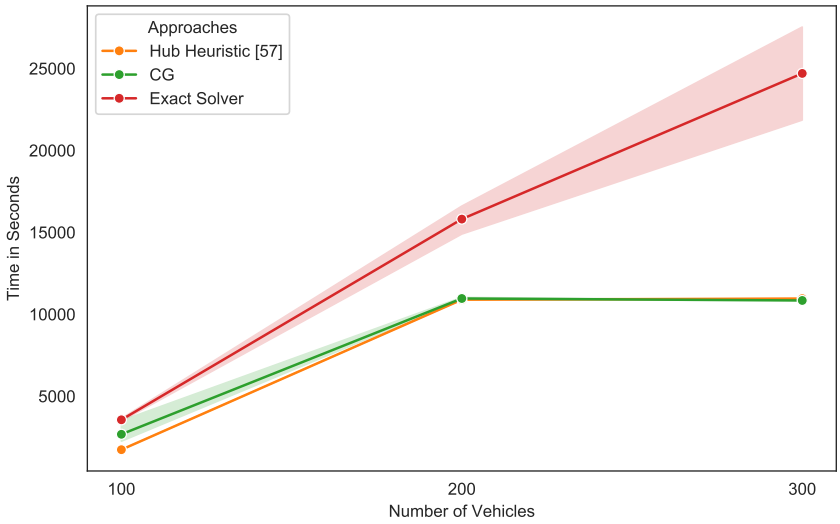


FIGURE 8.58: Execution times of the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 30×30 grid road network.

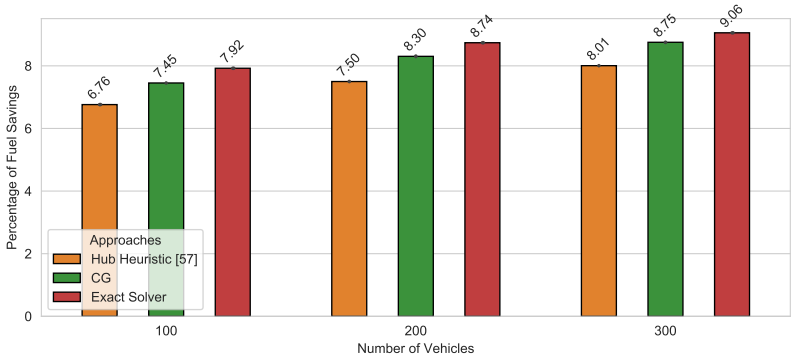


FIGURE 8.59: Fuel savings achieved by the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 100, 200, and 300 vehicles on the 30×30 grid road network.

TABLE 8.21: Execution times of **routing** algorithms in comparison to the baseline of the 30×30 Grid Road Network.

	Hub Heuristic		*Column Generation*		*Exact Solver*
	mean s	overh. %	mean s	overh. %	mean s
10	15.6587	446.92	1.59539	-44.2767	2.86306
20	41.0131	137.73	3.85484	-77.6556	17.2519
30	86.0089	-57.5934	28.596	-85.9008	202.82
50	277.692	-67.9684	453.169	-47.7274	866.933
100	1736.36	-51.239	2676.51	-24.8375	3560.96
200	10893	-31.1036	10964.4	-30.652	15810.7
300	10958.7	-55.6361	10844.5	-56.0984	24701.8

German Road Network Instance

Fig. 8.60 and Fig. 8.61 demonstrates performance of routing algorithms on the German road network. For the instance with 20 vehicles, *Hub Heuristic* achieves 0% savings within the time limit. In other words, the *Hub Heuristic* reaches the time limit during the initialization phase (see Lines 3 - 6); accordingly there is no solution available yet. Therefore, for the instances with more vehicles, the *Hub Heuristic* was excluded. The CG solves the problem instances with $10, 20, 30$, and 50 vehicles much more efficient than the *Exact Solver*. CG achieves good quality solutions for 20 and 30 instances. For the instance with 50 vehicles, the distance to the optimal solution increases. The computational results for $100, 200$, and 300 vehicles are also included in Fig. 8.60 and Fig. 8.61. The execution times of *Exact Solver* increase continuously with the number of vehicles. For the same vehicle instances, the CG algorithm shows better efficiency. Table 8.22 reports the execution times for all vehicle instances on the German road network. The column overh. % shows the calculation effort relative to the *Exact Solver* in percent. In the CG algorithm, we can observe negative overhead values, and these values show less computational effort in percent compared to *Exact Solver* computation time.

TABLE 8.22: Execution times of **routing** algorithms in comparison to the baseline of the German Road Network.

	Hub Heuristic		*Column Generation*		*Exact Solver*
	mean s	overh. $\%$	mean s	overh. $\%$	mean s
10	10105.6	6819.3	22.8476	-84.3562	146.049
20	10839.8	4624.96	40.2014	-82.4766	229.415
30	NaN	NaN	59.7469	-88.9722	541.782
50	NaN	NaN	136.952	-89.6992	1329.52
100	NaN	NaN	450.467	-76.271	1898.38
200	NaN	NaN	3162.07	-24.663	4197.23
300	NaN	NaN	2843.85	-59.7874	7072.04

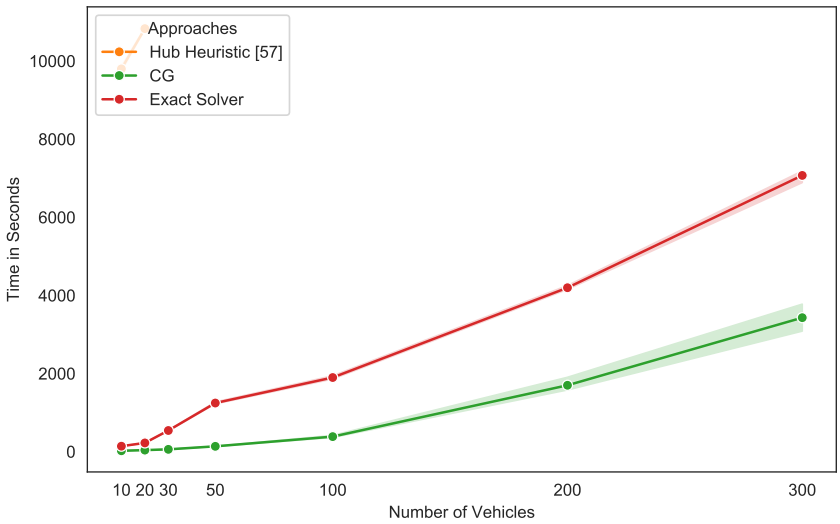


FIGURE 8.60: Execution times of the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, 50, 100, 200, and 300 vehicles on the German road network.

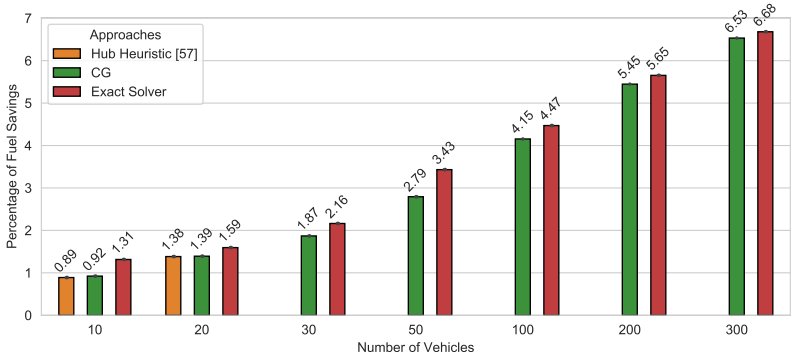


FIGURE 8.61: Fuel savings achieved by the routing algorithms: *Hub Heuristic*, CG, and *Exact Solver* for the instances with 10, 20, 30, 50, 100, 200, and 300 vehicles on the German road network.

Southern US Road Network Instance

Fig. 8.62 and Fig. 8.63 shows the performance of CG and *Exact Solver* on the Southern US road network. The execution time of *Exact Solver* explodes while the execution time of CG increases slightly. The CG algorithm achieves near-optimal solutions for $20, 30$, and 50 vehicles. For the instance with 100 vehicles, the *Exact Solver* runs out of memory. For the same instance, the CG algorithm shows excellent performance, with 3.42% savings. Table 5 provides a detailed overview of the execution times. Solving the instances with a large number of vehicles on vast road networks is a real challenge. The CG has excellent scalability characteristics and provides high-quality solutions for such problems.

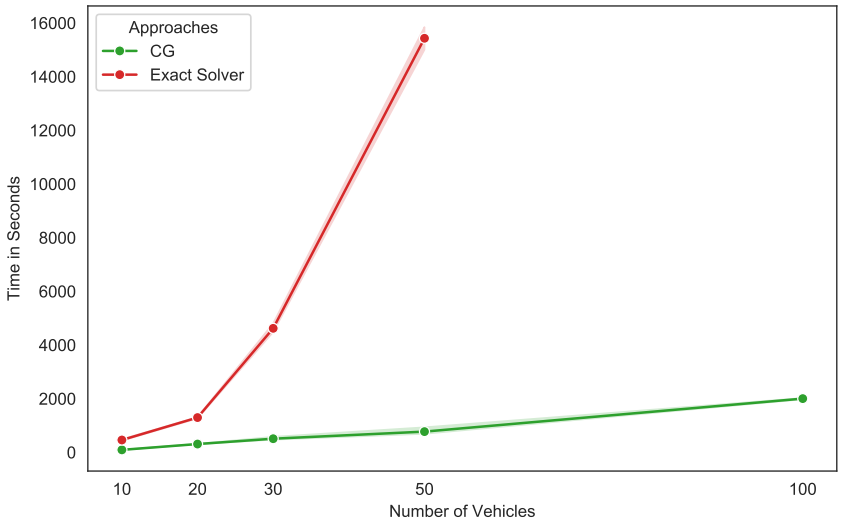


FIGURE 8.62: Execution times of the routing algorithms: CG and *Exact Solver* for the instances with 10, 20, 30, 50 and 100 vehicles on the Southern US road network.

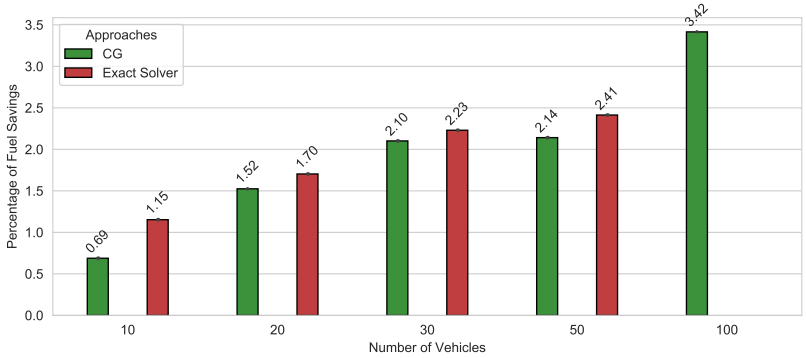


FIGURE 8.63: Fuel savings achieved by the routing algorithms: CG and *Exact Solver* for the instances with 10, 20, 30, 50 and 100 vehicles on the Southern US road network.

TABLE 8.23: Execution times of **routing** algorithms in comparison to baseline on the Southern US Road Network.

	Column Generation		*Exact Solver*
	mean s	overh. %	mean s
10	105.422	-77.2249	462.884
20	352.315	-73.1172	1310.56
30	823.69	-83.7471	5067.96
50	1491.83	-90.6976	16037
100	2004.08	NaN	NaN

8.3.1 Summarizing Experimental Results of Routing Algorithms

This section summarizes the experimental results of the routing algorithms. We compare related work algorithms *Best Pair* and *Hub Heuristic* [57] with our efficient Column Generation algorithm. The *Exact Solver* acts as a control method, which provides optimal solutions. *Best Pair* algorithm shows poor performance and reaches a time limit even for the simplest road networks, namely 10×10 Grid, 20×20 Grid, and Berlin networks. This makes the *Best Pair* algorithm not applicable to real-world problems. Compared with the *Best Pair*, *Hub Heuristic* exhibits better scalability. The *Hub Heuristic* determines solutions without reaching the time limit for the instances with $10, 20, 30$, and 50 vehicles on the following road networks: 10×10 Grid, 20×20 Grid, Berlin, Bavarian, Swedish and 30×30 Grid. The solutions discovered only are partially high quality. For the instances with a large number of vehicles, the *Hub Heuristic* reaches in most cases the time limit of 10800 seconds. For the German road network, the *Hub Heuristic* reaches the time limit already for 20 vehicles and achieves 0% savings. This makes it difficult to use the *Hub Heuristic* for the larger problem instances since it requires many resources. Our CG routing algorithm solves most problem instances more efficiently than the *Hub Heuristic*. In addition, the CG algorithm reaches the time limit for solving only for the instances with 200 and 300 vehicles on the road network 30×30 Grid. The *Exact Solver* also needs more time than expected to solve this instance. Fig. 8.64 shows the detailed comparison and convergence behavior of CG and *Exact Solver* for the instance with 200 vehicles on the 30×30 Grid road network. The blue plot represents the CG objective value as a function of time. The objective value is minimized over time according to the formulation in Eq. 6.1. The CG algorithm obtains a high-quality solution after ≈ 1400 seconds, and afterward, no significant improvement of the solution could be made. Finally, CG reaches the time limit within 10800 seconds without optimality. The green line marks the objective value of the optimal solution. First, the *Exact Solver* solves the relaxed ILP of the original problem, which is formulated in Eq. 2.8 - Eq. 2.12, with a dual simplex algorithm. The plot graph illustrates the objective values of performed dual simplex iterations. The objective values determined by the dual simplex are the lower bounds or **best bounds** of the primal problem. The **rate of convergence** of dual simplex slows down after 5700 seconds. For a more precise approximation, the optimal objective value algorithm takes an additional 9000 seconds. Solving the root relaxation is quite expensive, however, the provided result satisfies most integrality restrictions.

The execution time of CG is limited to 10800 seconds by default. The new stopping condition can be defined as follow: If no significant improvement can be found after a certain number of iterations, then terminate the algorithm. Further strategies can be developed to get closer to the optimal solution. The subproblem can generate, for example, reasonable alternative paths for a given vehicle [1]. The biggest advantage of CG is that many tailored strategies can be formulated, which can be combined or used at different stages of the solving process. Our CG algorithm solves the large problem instances very efficiently; nevertheless, the algorithm has much upward potential.

Statistical Analysis of Routing Algorithms in Terms of Execution Times

This section describes the statistical analysis of *Best Pair*, *Hub Heuristic*, CG, and *Exact Solver* algorithms. The road networks are divided into three different difficulty classes, as in Section 8.2.1. The first test considers 10×10 grid, 20×20 grid, Berlin, Bavarian, and Swedish road networks. Table 8.24 shows the results of *Friedman's* test with the Bergmann-Hommel post hoc procedure. The corresponding null hypothesis H_0 states that there is no difference between the routing algorithms. For the majority of algorithm pairs, the adjusted p-values are 0, and the corresponding H_0 is rejected. The p-value of CG and *Exact Solver* pair is > 0.05, and the corresponding H_0 is confirmed. In other words, the routing algorithms **differ significantly** in terms of execution time, except for the CG and *Exact Solver* algorithms, there is **no significant** difference. As already discussed, the *Best Pair* algorithm shows the **poorest** performance even for the simplest instances. In the instances with a large number of vehicles, the *Best Pair* reaches the time limit in any case. The *Hub Heuristic* is less efficient when compared to the CG and *Exact Solver* algorithms, especially for the instances with a large number of vehicles. Finally, the CG and *Exact Solver* showed the best performance for the

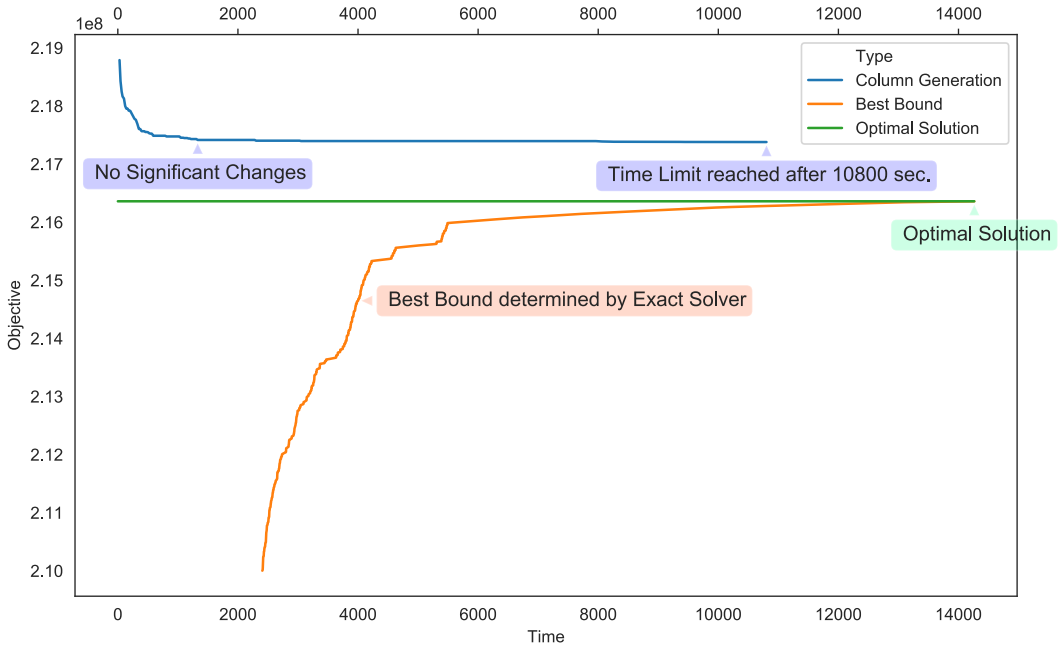


FIGURE 8.64: Comparison of the CG iterations and the dual simplex iterations of the Exact Solver.

simpler road networks. For the 10×10 grid and 20×20 grid road networks, the *Hub Heuristic* and CG algorithms provide average quality solutions. Furthermore, the approaches CG and *Hub Heuristic* achieve high-quality solutions for the Berlin, Bavarian, and Swedish road networks.

TABLE 8.24: Adjusted p-values of the Friedman test with Bergmann-Hommel post hoc procedure [31, 45, 21]. The *Best Pair*, CG, *Hub Heuristic*, and *Exact Solver* algorithms are compared with each other in the test. The test has been executed for 10, 20, 30, 50, 100, 200, and 300 vehicles instances on 10×10 grid, 20×20 grid, Berlin, Bavarian, and Swedish road networks. The results of the test shows significant statistical differences among the **routing** algorithms with a level of significance at $\alpha = 0.05$.

	Best Pair	CG	*Hub Heuristic*	*Exact Solver*
Best Pair		0.000	0.000	0.000
CG	0.000		0.000	0.456
Hub Heuristic	0.000	0.000		0.000
Exact Solver	0.000	0.456	0.000	

For the German and Southern US road networks, the CG solves the problem instances much more efficiently than the *Exact Solver*. The CG achieves near-optimal solutions and uses considerably less resources. For the Southern US road network, the use of CG is particularly important since *Exact Solver*'s execution time increases very rapidly.

8.4 Experiments of Grouping-based CG Routing Algorithm

The experimental results in Section 8.2 and Section 8.3 confirm that heuristic approaches are necessary to solve the instances where the large road networks are involved. This section evaluates the performance of grouping approaches combined with a CG routing algorithm to solve instances with a very large number > 300 of vehicles on the Southern US road network.

TABLE 8.25: Execution times of incentives, grouping, and routing algorithms on the Southern US Road Network.

$\|H\|$	Grouping Alg.	Incentives Times (s)	Grouping Times (s)	Routing Times (s)	Total Times (s)
100	*Greedy CPP*	54.195603	0.236594	244.965820	299.400375
	SGVNS CPP	54.920206	1.282616	262.106066	318.312020
200	*Greedy CPP*	218.988676	0.818121	568.281866	788.091806
	SGVNS CPP	256.085926	9.020369	464.758257	729.868107
300	*Greedy CPP*	487.304061	1.820104	1043.371651	1532.500446
	SGVNS CPP	491.147957	28.483530	937.850616	1457.486627
500	*Greedy CPP*	1360.782055	6.241779	2161.176335	3528.208054
	SGVNS CPP	1343.242561	270.134792	1774.272403	3387.657223
700	*Greedy CPP*	2614.484217	15.925033	2987.233105	5617.653050
	SGVNS CPP	2647.154158	727.378606	2476.277473	5850.820626
1000	*Greedy CPP*	5481.518556	45.649916	16090.888362	21618.072306
	SGVNS CPP	5385.081234	2263.242319	5134.411370	12782.751485
1500	*Greedy CPP*	12224.833309	562.706476	31140.099526	43927.665601
	SGVNS CPP	12393.190414	6075.182800	19613.795336	38082.194067
2000	*Greedy CPP*	22429.379780	4859.419219	31095.517793	58384.353607
	SGVNS CPP	22283.898879	24464.487006	28517.750259	75266.171271

Fig. 8.65 and Fig. 8.66 illustrates the experimental results with 100, 200, 300, 500, 700, 1000, 1500, and 2000 vehicles on the Southern US road network. Such extensive problem instances can only be solved with a combination of proposed grouping approaches and CG algorithm. The *Greedy CPP* and *SGVNS CPP* approaches have similar performance for the instances with 100, 200, 300, 500, and 700 vehicles. For the 1000 and 1500 vehicles, the *Greedy CPP* approach is more performant than the *SGVNS CPP* approach. For 2000 vehicles, the *Greedy CPP* shows better time performance compared to *SGVNS CPP*. Table 8.25 provides a detailed overview of the execution times. The grouping times of *SGVNS CPP* increase sharply for the instances with 1000, 1500, and 2000 vehicles. This issue can be improved by optimizing the implementation and adjusting the $\tilde{\beta}$ parameter, which controls the distance between the solutions, see Section 5.2.2. The routing times of *SGVNS CPP* is shorter in contrast to *Greedy CPP*.

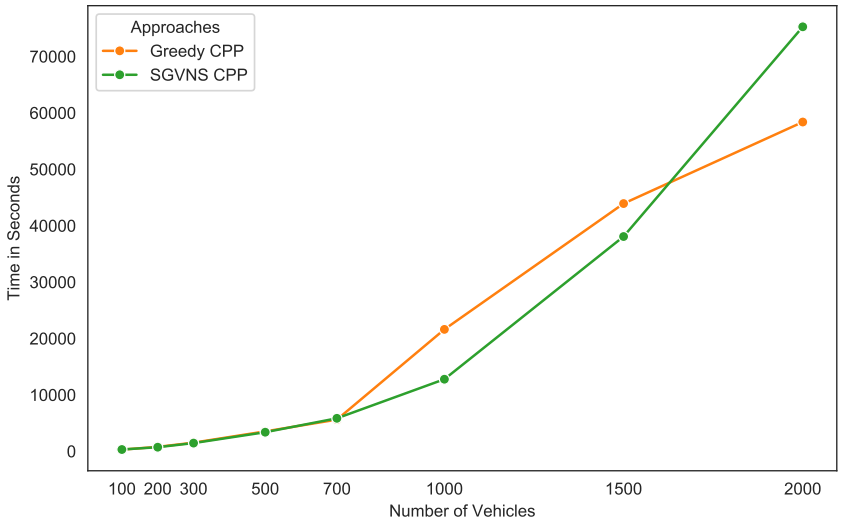


FIGURE 8.65: Execution times of the grouping-based routing algorithms: *α-cut Method*, *Greedy CPP*, and *SGVNS CPP* for the instances with 100, 200, 300, 500, 700, 1000, 1500, and 2000 vehicles on the Southern US road network.

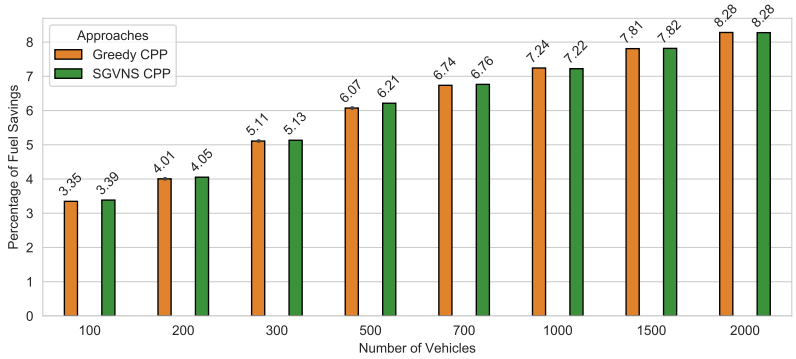


FIGURE 8.66: Fuel savings achieved by the grouping-based routing algorithms for the instances with 100, 200, 300, 500, 700, 1000, 1500, and 2000 vehicles on the Southern US road network.

8.5 Pilot Experiments and the Determination of Initial Parameters

This section briefly describes the parameters used in the experiments of this work. The parameters are determined by pilot experiments and by the heuristic evaluation.

8.5.1 Parameters of the Incentive Methods

The linear platooning saving parameter η' is set to 0.5 by default, see Section 5.1.2. Eq. 5.2 weighs and summarizes the incentives of the different methods into a single value, and the weights are uniformly distributed.

8.5.2 Penalty Factor for the Incentive Graph $\tilde{G}$.

Basically the vehicles with zero incentives must be assigned into different groups. We can replace zero incentives with a large negative number $M = -1 \cdot |H| \cdot |E|$ to avoid wrong grouping. In this case, the corresponding vehicles will be assigned to **different** groups. However, if we set a very small negative number, e.g. $M = -0.1$, then the probability increases, and the vehicles are assigned into the same group. In Eq. 8.3 we define a penalty function, which allows to parameterizing the M number by penalty factor $\tilde{p}$. The function takes into account the number of vehicles and was determined by trial and error.

$$\tilde{P}(\tilde{p}) = -1 \cdot \frac{|H| \cdot |H| \cdot \tilde{p}}{50} \tag{8.3}$$

TABLE 8.26: The pilot testing to determine pareto-optimal penalty factor for the grouping-based routing algorithms. The German road network instance with 100 vehicles is used.

$\|H\|$	$\tilde{p}$	Times (s)	Savings %
100	0.0001	1213.705106	4.375403
	0.0005	696.423114	4.306264
	0.001	599.583388	4.318286
	0.005	466.311807	4.132101

8.5.3 α-*cut Method* Parameters.

α-cut Method requires an initial $\tilde{\alpha}$ or $\tilde{k}$ parameter, and this is the biggest disadvantage. We use in our experiments a $\tilde{k}$ parameter, which specifies the number of vehicle groups. The $\tilde{k}$ parameter is selected depending on the number of vehicles $\tilde{k} = |H| \cdot 2$. It should be noted that a low value may cause poor results and a high value long calculation times. Further experiments are necessary to determine the full potential.

8.6 Summary

This chapter described the experiments of this work in great detail. The definition of the experiments is outlined in Section 8.1. It introduces the problem instances, experiment methodologies, and visualization methodologies. Section 8.2 provides a detailed experimental report of our grouping-based routing algorithms. The approaches were tested with different problem instances and compared to each other. Section 8.2.1 summaries and analyses statistically the experimental results. Section 8.3 reports the performance comparisons of routing algorithms. We evaluated the state-of-the-art algorithms and compared them with our novel CG algorithm. Section 8.4 evaluates the extremely large problem instances with up to 2000 vehicles on the Southern US road network.

Chapter 9

Conclusions and Future Research

In this chapter, the primary outcomes of this thesis are summarized, and conclusions are drawn. Furthermore, some future research directions will be suggested.

9.1 Outcomes and Conclusions

Platooning is the coupling of two or more trucks with small inter-vehicle distance. This technology helps to improve road safety, increases road capacity, and reduces CO_2 emissions. This thesis focuses on minimizing the overall fuel consumption by the centralized formation of vehicles into platoons. **The overall goal of this thesis** is to propose and evaluate novel efficient and effective algorithms for the fuel-optimal vehicle platooning problem that copes well with large-scale data-intensive problem instances. **This goal has been achieved successfully.**

We conducted an extensive experimental evaluation of our proposed algorithms and compared them against the state-of-the-art algorithms from the literature and exact solvers for integer linear programming as the baseline. For our experiments, we used eight road network instances with different sizes and topologies. The road networks included four synthetically generated grid networks as well as four real-world road networks. From the large road network instances, we derived large-scale problem instances with many vehicles (up to 2000 for some of the experiments). To the best of our knowledge, state-of-the-art approaches for vehicle platooning that were proposed in the literature only work well for small- or medium-size problem instances. The principal motivation for our thesis was to develop new innovative strategies for tackling large-scale problem instances.

To address our Objective (1), we investigated our approach of first grouping vehicles with high platooning incentives together and subsequent computation of the platooning routes for each of these promising groups. We discussed novel strategies for detecting promising vehicle groups. Our approach for defining and computing incentives delivers information about the platooning opportunities with low computational effort. The computation time of the incentive calculation depends on the number of vehicles, but not on the size of the road network. These methods can be flexibly combined, weighted, and extended for the specific cases. The incentives have a direct impact on the grouping process and accordingly on the quality of the platoons. A by-product of the incentives computation is the vehicle geometric containers, which we exploited to reduce the region of the road network that needs to be considered for fuel-optimal platooning. This significantly accelerates performance and is an important contribution. We conclude that our incentives methods are a useful tool to cope with large-size road networks. It should be emphasized that the incentive methods are easily parallelizable so that efficiency can be improved further.

Based on the platooning incentives, we introduced different vehicle grouping algorithms in order to detect vehicle groups with high platooning potential. The resulting groups represent subproblems of the original problem that can be solved independently of each other. This thesis presents four different grouping approaches, which deliver vehicle groups with different structures. The *α-cut Method* finds non-disjoint vehicle groups with a desirable number of groups. These can then be resolved to make them mutually disjoint. In our experiments, we observed that the *α-cut Method* efficiently delivers groups, even for a large number of vehicles. However, the quality of the groups is often far from optimal, especially for larger road networks.

To overcome this issue, we established a relationship between finding promising vehicle groups and finding clique partitions in graphs. Using this relationship, we proposed further graph-based grouping algorithms (*Exact CPP*, *Greedy CPP*, *SGVNS CPP*). Unfortunately, finding clique partitions is a hard problem, so we looked for heuristics too. *Exact CPP* delivers optimal or near-optimal solutions but shows very poor scalability. To improve the scalability of the grouping process, we proposed a greedy algorithm (*Greedy CPP*). During our experiments, we observed that *Greedy CPP* usually finds very few groups with many vehicles, while the remaining groups have only a small number of vehicles. *Greedy CPP* is an efficient grouping method to tackle large-scale problem instances but achieves significantly less fuel savings. *Skewed General Variable Neighborhood Search (SGVNS)* is based on a metaheuristic approach. This method provides desirable groups efficiently for a large number of vehicles. In our experiments, we observed that in case of a low spontaneous platooning rate, *SGVNS CPP* provides optimal or near-optimal solutions and achieves better results than the α*-cut Method* or *Greedy CPP*. We conclude that *SGVNS CPP* is the most suitable method for vehicle grouping in fuel-optimal platooning.

To address our Objective (2), we developed novel efficient algorithms for finding near-optimal routings of promising vehicle groups. First, we analyzed two state-of-the-art heuristic methods, *Best Pair* and *Hub Heuristic* [57], for fuel-optimal vehicle platooning. *Best Pair* has poor performance and reaches the time limit even for the smallest road networks. This makes *Best Pair* not suitable for real-world problem instances. *Hub Heuristic* has better performance than *Best Pair* and, in some cases, outperforms the baseline (*Exact Solver*). *Hub Heuristic* can solve the instances with a large number of vehicles within a time limit. In the literature [57], the state-of-the-art approaches were only evaluated with much smaller road networks and smaller numbers of vehicles. Thus, for larger problem instances, no information on the computation time is available.

To overcome this situation, we proposed a novel Column Generation formulation for fuel-optimal vehicle platooning. We provided an equivalent path-based formulation for the master problem based on the common integer linear programming model for vehicle platooning. Herein, the number of variables is much larger than the number of constraints, which is beneficial for applying the Column Generation technique. Using the Column Generation technique, we proposed a new algorithm that restricts and solves the master problem iteratively. The corresponding subproblems generate new promising platooning paths and add these to the restricted master problem. The restricted master problem can be solved very efficiently in our algorithm since only a small subset of variables is considered at a time. Thus, the computation time for finding near-optimal solutions can be reduced considerably. Our proposed CG algorithm can solve large-scale problem instances efficiently. Our experiments show that our proposed CG algorithm outperforms the baseline (*Exact Solver*), especially for problem instances with large road networks. Our proposed CG algorithm can tackle road networks with up to $300,000$ edges and up to 100 vehicles. It should be noted that the solution of the subproblems is parallelizable, which gives room for further improvement.

Afterward, we analyzed the combination of vehicle grouping and vehicle routing. In particular, we used our Column Generation algorithm to each of the promising vehicle groups obtained by our proposed grouping algorithms. Our experimental evaluation shows that this combination can compute solutions for large-scale problem instances with up to 300,000 road segments and 2000 vehicles. This is a significant improvement over the current state-of-the-art. **The most important conclusion of this thesis is that the decomposition of vehicle platooning by vehicle grouping is a high-performance strategy for tackling large-scale problem instances.**

Eventually, the grouping-based routing solutions can be time scheduled subsequently by our proposed scheduling algorithm that computes the travel times for predetermined platoon routings.

Another challenge when dealing with large-scale problem instances is the resulting data intensity. Our algorithms interact with several gigabytes of problem-specific data, such as road networks, vehicles and vehicle assignments, platooning incentives, vehicle groups, shortest paths, platooning paths, and the corresponding index structures. **To address our Objective (3)**, we developed a novel property graph data model to capture all platooning

relevant data. We built an innovative graph database upon our efficient property graph data model to store and manage the large volumes of platooning relevant data that are characteristic for large-scale problem instances. Additionally, we use our novel filter-based modeling rule [89] to accelerate query performance further. Even for the largest of our road networks and up to 2000 vehicles, our proposed algorithms show good performance when querying and updating data in the database.

In summary, the combination of our vehicle grouping algorithms, our CG vehicle routing algorithm and our use of innovative graph database technologies **enable us to solve very large problem instances with up to 2000 vehicles on road networks with up to 300,000 edges**.

9.2 Future Work

This section makes several suggestions for further improvement and future work directions. One obvious idea is to extend our work to vehicle platooning with time constraints, e.g., earliest departure times and latest arrival time. Such time constraints could be integrated into the proposed incentives algorithms. The vehicle combinations that violate the time restrictions can be discarded early on, e.g., by assigning them high penalty costs. A third dimension can extend the proposed incentive methods for the time similar to [71]. Once this is done, the vehicle groups can then be formed as usual using the grouping algorithms proposed in this thesis.

This thesis proposes the first Column Generation formulation of fuel-optimal vehicle platooning. The analysis of our experiments showed the slow convergence of solutions for some problem instances. This effect is known as the long tail effect and is caused by the optimization problem's degeneracy [58]. In the literature, several approaches are given to stabilize CG-based algorithms [3, 26, 77, 81]. Time restrictions and variable vehicle speeds should carefully extend the current CG formulation. However, this strategy can be computationally expensive.

Subsequently, several computational approaches, each with a set of parameters, are proposed in this thesis. Selecting the best approaches and choosing suitable parameters is not a trivial task. A dynamic approach could help to determine the appropriate configuration for given problem instances. For this purpose, machine learning methods could be used.

Bibliography

[1] Ittai Abraham et al. "Alternative routes in road networks". In: *Journal of Experimental Algorithmics (JEA)* 18 (2013), pp. 1–3.

[2] Akanksha Agrawal et al. "Kernelization of cycle packing with relaxed disjointness constraints". In: *LIPIcs-Leibniz International Proceedings in Informatics*. Vol. 55. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016.

[3] Hatem Ben Amor, Jacques Desrosiers, and Antonio Frangioni. *Stabilization in column generation*. Groupe d'études et de recherche en analyse des décisions, 2004.

[4] Lakshmi Dhevi Baskar, Bart De Schutter, and Hans Hellendoorn. "Optimal routing for intelligent vehicle highway systems using mixed integer linear programming". In: *IFAC Proceedings Volumes* 42.15 (2009), pp. 569–575.

[5] Hans L Bodlaender, Stéphan Thomassé, and Anders Yeo. "Kernel bounds for disjoint cycles and disjoint paths". In: *Theoretical Computer Science* 412.35 (2011), pp. 4570–4578.

[6] Paul T Boggs and Jon W Tolle. "Sequential quadratic programming". In: *Acta numerica* 4 (1995), pp. 1–51.

[7] Christophe Bonnet and Hans Fritz. *Fuel consumption reduction in a platoon: Experimental results with two electronically coupled trucks at close spacing*. Tech. rep. SAE Technical Paper, 2000.

[8] Jack Brimberg, Nenad Mladenović, and Dragan Urošević. "Solving the maximally diverse grouping problem by skewed general variable neighborhood search". In: *Information Sciences* 295 (2015), pp. 650–675.

[9] Jack Brimberg et al. "Solving the clique partitioning problem as a maximally diverse grouping problem". In: *Optimization Letters* 11.6 (2017), pp. 1123–1135.

[10] Coen Bron and Joep Kerbosch. "Algorithm 457: finding all cliques of an undirected graph". In: *Communications of the ACM* 16.9 (1973), pp. 575–577.

[11] Fred Browand, John McArthur, and Charles Radovich. "Fuel saving achieved in the field test of two tandem trucks". In: (2004).

[12] Michael J Brusco and Hans-Friedrich Köhn. "Clustering qualitative data based on binary equivalence relations: neighborhood search heuristics for the clique partitioning problem". In: *Psychometrika* 74.4 (2009), p. 685.

[13] Aydın Buluç et al. "Recent advances in graph partitioning". In: *Algorithm Engineering*. Springer, 2016, pp. 117–158.

[14] Gerrit Burmester. "Ein Lösungsansatz für das Vehicle Platooning Problem durch den Einsatz der Spaltengenerierungsmethode". In: Clausthal University of Technology - Institute of Computer Sciences. 2018.

[15] *Business Intelligence and Analytics Software*. `https://www.tableau.com/`. (Accessed on 03/29/2020).

[16] Jason Carbaugh, Datta N Godbole, and Raja Sengupta. "Safety and capacity analysis of automated and manual highway systems". In: *Transportation Research Part C: Emerging Technologies* 6.1-2 (1998), pp. 69–99.

[17] Eunjeong Choi and Dong-Wan Tcha. "A column generation approach to the heterogeneous fleet vehicle routing problem". In: *Computers & Operations Research* 34.7 (2007), pp. 2080–2095.

[18] Saul G De Amorim, Jean-Pierre Barthélemy, and Celso C Ribeiro. "Clustering and clique partitioning: simulated annealing and tabu search approaches". In: *Journal of Classification* 9.1 (1992), pp. 17–41.

[19] Ayhan Demiriz, Kristin P Bennett, and John Shawe-Taylor. "Linear programming boosting via column generation". In: *Machine Learning* 46.1-3 (2002), pp. 225–254.

[20] John E Dennis Jr and Jorge J Moré. "Quasi-Newton methods, motivation and theory". In: *SIAM review* 19.1 (1977), pp. 46–89.

[21] Joaquín Derrac et al. "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms". In: *Swarm and Evolutionary Computation* 1.1 (2011), pp. 3–18.

[22] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*. Vol. 5. Springer Science & Business Media, 2006.

[23] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. "A new optimization algorithm for the vehicle routing problem with time windows". In: *Operations research* 40.2 (1992), pp. 342–354.

[24] *DHL Facilities*. URL: https://hifld-geoplatform.opendata.arcgis.com.

[25] Stuart E Dreyfus and Robert A Wagner. "The Steiner problem in graphs". In: *Networks* 1.3 (1971), pp. 195–207.

[26] Olivier Du Merle et al. "Stabilized column generation". In: *Discrete Mathematics* 194.1-3 (1999), pp. 229–237.

[27] Reeves Fletcher and Colin M Reeves. "Function minimization by conjugate gradients". In: *The computer journal* 7.2 (1964), pp. 149–154.

[28] International Transport Forum. *ITF Transport Outlook 2017*. 2017, p. 224. DOI: https://doi.org/https://doi.org/10.1787/9789282108000-en. URL: https://www.oecd-ilibrary.org/content/publication/9789282108000-en.

[29] *Frachtzentren und Paketzentren von DHL*. URL: https://www.paketda.de/paketdepot-dhl.html.

[30] *framework for running and analyzing computational experiments.* URL: https://pascalkleindienst.github.io/experimentum/.

[31] Milton Friedman. "A comparison of alternative tests of significance for the problem of m rankings". In: *The Annals of Mathematical Statistics* 11.1 (1940), pp. 86–92.

[32] Anastasia Frolov. "Business Intelligence - Implementing of Improved Fleet Sales Reporting within the Volkswagen Group by using the Analysis and Reporting System QLIKVIEW". In: Clausthal University of Technology - Institute of Computer Sciences. 2017.

[33] Gyozo Gidofalvi et al. "Highly scalable trip grouping for large-scale collective transportation systems". In: *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*. ACM. 2008, pp. 678–689.

[34] Renata Gnatowska and Marcin Sosnowski. "The influence of distance between vehicles in platoon on aerodynamic parameters". In: *EPJ Web of Conferences*. Vol. 180. EDP Sciences. 2018, p. 02030.

[35] Franz Graf et al. "MARiO: multi-attribute routing in open street map". In: *International Symposium on Spatial and Temporal Databases*. Springer. 2011, pp. 486–490.

[36] Martin Grötschel and Yoshiko Wakabayashi. "A cutting plane algorithm for a clustering problem". In: *Mathematical Programming* 45.1-3 (1989), pp. 59–96.

[37] *Gurobi Optimizer*. `https://www.gurobi.com/de/resource/mip-basics/`. (Accessed on 17/02/2020).

[38] *Gurobi Optimizer - MIP Models*. `https://www.gurobi.com/documentation/9.0/refman/mip_models.html`. (Accessed on 17/02/2020).

[39] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching*. Vol. 14. 2. ACM, 1984.

[40] Pierre Hansen and Nenad Mladenović. "An introduction to variable neighborhood search". In: *Meta-heuristics*. Springer, 1999, pp. 433–458.

[41] Pierre Hansen, Nenad Mladenović, and José A Moreno Pérez. "Variable neighbourhood search: methods and applications". In: *4OR* 6.4 (2008), pp. 319–360.

[42] Sebastian Van de Hoef. "Fuel-efficient centralized coordination of truck platooning". PhD thesis. KTH Royal Institute of Technology, 2016.

[43] Sebastian van de Hoef, Karl H Johansson, and Dimos V Dimarogonas. "Computing feasible vehicle platooning opportunities for transport assignments". In: *IFAC-PapersOnLine* 49.3 (2016), pp. 43–48.

[44] Sebastian van de Hoef, Karl Henrik Johansson, and Dimos V Dimarogonas. "Fuel-efficient en route formation of truck platoons". In: *IEEE Transactions on Intelligent Transportation Systems* 19.1 (2017), pp. 102–112.

[45] Gerhard Hommel and Gudrun Bernhard. "Multiple hypotheses testing". In: *Computational Aspects of Model Choice*. Springer, 1993, pp. 211–235.

[46] Vaida Jakoniene, David Rundqvist, and Patrick Lambrix. "A method for similarity-based grouping of biological data". In: *Lecture Notes in Computer Science* 4075 (2006), p. 136.

[47] A Hamish Jamson et al. "Behavioural changes in drivers experiencing highly-automated vehicle control in varying traffic conditions". In: *Transportation research part C: emerging technologies* 30 (2013), pp. 116–125.

[48] Seiichi Kagaya, Shinya Kikuchi, and Robert A Donnelly. "Use of a fuzzy theory technique for grouping of trips in the vehicle routing and scheduling problem". In: *European Journal of Operational Research* 76.1 (1994), pp. 143–154.

[49] Brian Kallehauge et al. "Vehicle routing problem with time windows". In: *Column generation*. Springer, 2005, pp. 67–98.

[50] Christoph Kammer. *Coordinated heavy truck platoon routing using global and locally distributed approaches*. 2013.

[51] Shinya Kikuchi and Natasa Vukadinovic. "Grouping trips by fuzzy similarity for scheduling of demand-responsive transportation vehicles". In: *Transportation Planning and Technology* 18.1 (1994), pp. 65–80.

[52] M Lammert and J Gonder. *Reducing Fuel Consumption through Semi-Automated Platooning with Class 8 Tractor Trailer Combinations (Poster)*. Tech. rep. National Renewable Energy Lab.(NREL), Golden, CO (United States), 2014.

[53] Michael P Lammert et al. "Effect of platooning on fuel consumption of class 8 vehicles over a range of speeds, following distances, and mass". In: *SAE International Journal of Commercial Vehicles* 7.2014-01-2438 (2014), pp. 626–639.

[54] Jeffrey Larson, Kuo-Yun Liang, and Karl H Johansson. "A distributed framework for coordinated heavy-duty vehicle platooning". In: *IEEE Transactions on Intelligent Transportation Systems* 16.1 (2014), pp. 419–429.

[55] Jeffrey Larson, Todd Munson, and Vadim Sokolov. "Coordinated platoon routing in a metropolitan network". In: *2016 Proceedings of the Seventh SIAM Workshop on Combinatorial Scientific Computing*. SIAM. 2016, pp. 73–82.

[56] Jeffrey Larson et al. "Coordinated route optimization for heavy-duty vehicle platoons". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. IEEE. 2013, pp. 1196–1202.

[57] Erik Larsson, Gustav Sennton, and Jeffrey Larson. "The vehicle platooning problem: Computational complexity and heuristics". In: *Transportation Research Part C: Emerging Technologies* 60 (2015), pp. 258–277.

[58] Chungmok Lee and Sungsoo Park. "Chebyshev center based column generation". In: *Discrete Applied Mathematics* 159.18 (2011), pp. 2251–2265.

[59] Kuo-Yun Liang, Jonas Mårtensson, and Karl H Johansson. "Heavy-duty vehicle platoon formation for fuel efficiency". In: *IEEE Transactions on Intelligent Transportation Systems* 17.4 (2015), pp. 1051–1061.

[60] Kuo-Yun Liang, Jonas Mårtensson, and Karl Henrik Johansson. "When is it fuel efficient for a heavy duty vehicle to catch up with a platoon?" In: *IFAC Proceedings Volumes* 46.21 (2013), pp. 738–743.

[61] Dennis Luxen and Dennis Schieferdecker. "Candidate sets for alternative routes in road networks". In: *International Symposium on Experimental Algorithms*. Springer. 2012, pp. 260–270.

[62] Rita Macedo et al. "Skewed general variable neighborhood search for the location routing scheduling problem". In: *Computers & Operations Research* 61 (2015), pp. 143–152.

[63] *Matplotlib: Python plotting - Matplotlib 3.2.1 documentation*. `https://matplotlib.org/`. (Accessed on 03/29/2020).

[64] *Mixed-Integer Programming (MIP) - A Primer on the Basics - Gurobi*. `https://www.gurobi.com/de/products/gurobi-optimizer/`. (Accessed on 01/02/2020).

[65] N. Mladenović and P. Hansen. "Variable Neighborhood Search". In: *Comput. Oper. Res.* 24.11 (Nov. 1997), pp. 1097–1100. ISSN: 0305-0548. DOI: 10.1016/S0305-0548(97)00031-2. URL: http://dx.doi.org/10.1016/S0305-0548(97)00031-2.

[66] Nenad Mladenović et al. "General variable neighborhood search for the continuous optimization". In: *European Journal of Operational Research* 191.3 (2008), pp. 753–770.

[67] Directorate-General for Mobility and Transport (European Commission). *EU Transport in Figures: Statistical Pocketbook 2019*. 2019.

[68] *Neo4j Graph Platform - Memory configuration*. `https://neo4j.com/docs/operations-manual/current/performance/memory-configuration/`. (Accessed on 17/02/2020).

[69] *Neo4j Graph Platform - Periodic Execution*. `https://neo4j.com/docs/labs/apoc/current/graph-updates/periodic-execution/`. (Accessed on 17/02/2020).

[70] *Neo4j Graph Platform - The Leader in Graph Databases*. `https://neo4j.com/`. (Accessed on 17/02/2020).

[71] Tijs Neutens et al. "A three-dimensional network-based space–time prism". In: *Journal of Geographical Systems* 10.1 (2008), pp. 89–107.

[72] Abtin Nourmohammadzadeh and Sven Hartmann. "Fuel-efficient truck platooning by a novel meta-heuristic inspired from ant colony optimisation". In: *Soft Computing* 23.5 (2019), pp. 1439–1452.

[73] Abtin Nourmohammadzadeh and Sven Hartmann. "Fuel Efficient Truck Platooning with Time Restrictions and Multiple Speeds Solved by a Particle Swarm Optimisation". In: *International Conference on Theory and Practice of Natural Computing*. Springer. 2018, pp. 188–200.

[74] Abtin Nourmohammadzadeh and Sven Hartmann. "The fuel-efficient platooning of heavy duty vehicles by mathematical programming and genetic algorithm". In: *International Conference on Theory and Practice of Natural Computing*. Springer. 2016, pp. 46–57.

[75] Maarten Oosten, Jeroen HGC Rutten, and Frits CR Spieksma. "The clique partitioning problem: facets and patching facets". In: *Networks* 38.4 (2001), pp. 209–226.

[76] *OpenStreetMap*. URL: `https://www.openstreetmap.org/`.

[77] Amar Oukil et al. "Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems". In: *Computers & Operations Research* 34.3 (2007), pp. 817–834.

[78] *pandas - Python Data Analysis Library*. `https://pandas.pydata.org/`. (Accessed on 03/29/2020).

[79] PascalKleindienst. *Experimentum - Project Code at GitHub*. 2019. URL: `https://github.com/PascalKleindienst/experimentum/tree/master/example`.

[80] Xinwu Qian et al. "Optimal assignment and incentive design in the taxi group ride problem". In: *Transportation Research Part B: Methodological* (2017).

[81] Louis-Martin Rousseau, Michel Gendreau, and Dominique Feillet. "Interior point stabilization for column generation". In: *Operations Research Letters* 35.5 (2007), pp. 660–668.

[82] Siddhartha Sahu et al. "The ubiquity of large graphs and surprising challenges of graph processing". In: *Proceedings of the VLDB Endowment* 11.4 (2017), pp. 420–431.

[83] Jesús Sánchez-Oro, Nenad Mladenović, and Abraham Duarte. "General variable neighborhood search for computing graph separators". In: *Optimization Letters* 11.6 (2017), pp. 1069–1089.

[84] *seaborn:; statisical data visualization - seaborn 0.10.0 documentation*. `https://seaborn.pydata.org/`. (Accessed on 03/29/2020).

[85] Jeff Smith et al. "Aerodynamic impact of tractor-trailer in drafting configuration". In: *SAE International Journal of Commercial Vehicles* 7.2014-01-2436 (2014), pp. 619–625.

[86] Vadim Sokolov et al. "Platoon formation maximization through centralized routing and departure time coordination". In: *arXiv preprint arXiv:1701.01391* (2017).

[87] Marius M Solomon. "Algorithms for the vehicle routing and scheduling problems with time window constraints". In: *Operations research* 35.2 (1987), pp. 254–265.

[88] Dietrich Steinmetz, Gerrit Burmester, and Sven Hartmann. "A Fast Heuristic for Finding Near-Optimal Groups for Vehicle Platooning in Road Networks". In: *International Conference on Database and Expert Systems Applications*. Springer. 2017, pp. 395–405.

[89] Dietrich Steinmetz et al. "A Graph Model for Taxi Ride Sharing Supported by Graph Databases". In: *International Conference on Conceptual Modeling*. Springer. 2019, pp. 108–116.

[90] Dietrich Steinmetz et al. "Using a Conceptual Model to Transform Road Networks from OpenStreetMap to a Graph Database". In: *International Conference on Conceptual Modeling*. Springer. 2018, pp. 301–315.

[91] *Trucks on a European tour for platooning*. URL: `https://www.volvogroup.com/en-en/news/2016/mar/news-151620.html`.

[92] Sebastian Van De Hoef, Karl H Johansson, and Dimos V Dimarogonas. "Fuel-optimal centralized coordination of truck platooning based on shortest paths". In: *2015 American Control Conference (ACC)*. IEEE. 2015, pp. 3740–3745.

[93] Sebastian Van De Hoef, Karl Henrik Johansson, and Dimos V Dimarogonas. "Coordinating truck platooning by clustering pairwise fuel-optimal plans". In: *Intelligent Transportation Systems (ITSC), 2015 IEEE 18th International Conference on*. IEEE. 2015, pp. 408–415.

[94] Prasad Vegendla et al. *Investigation of aerodynamic influence on truck platooning*. Tech. rep. SAE Technical Paper, 2015.

[95] Chad Vicknair et al. "A comparison of a graph database and a relational database: a data provenance perspective". In: *Proceedings of the 48th annual Southeast regional conference*. ACM. 2010, p. 42.

[96] Dorothea Wagner, Thomas Willhalm, and Christos Zaroliagis. "Geometric containers for efficient shortest-path computation". In: *Journal of Experimental Algorithmics (JEA)* 10 (2005), pp. 1–3.

[97] Haibo Wang et al. "Solving group technology problems via clique partitioning". In: *International Journal of Flexible Manufacturing Systems* 18.2 (2006), pp. 77–97.

[98] Da Yan, Zhou Zhao, and Wilfred Ng. "Efficient algorithms for finding optimal meeting point on road networks". In: *Proceedings of the VLDB Endowment* 4.11 (2011), pp. 1–11.

[99] Wei Zhang, Marcus Sundberg, and Anders Karlström. "Platoon coordination with time windows: an operational perspective". In: *Transportation Research Procedia* 27 (2017), pp. 357–364.

[100] Julia Zillies et al. "A column generation approach for optimized routing and coordination of a UAV fleet". In: *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE. 2016, pp. 350–357.

List of Figures

List of Tables

Abbreviations

MP Master Problem

SP Subproblem

RMP Restricted Master Problem

VGCC Vehicle Geometric Convex Container

VGC Vehicle Geometric Container

I VGCC Intersection

ITF International Transport Forum

ILP Integer Linear Program

MIP Mixed Integer Programming

CPP Clique Partitioning Problem

NP (non-deterministic polynomial-time

OSM OpenStreetMap

VND Variable Neighborhood Descent

VNS Variable Neighborhood Search

VPP Vehicle Platooning Problem

VRP Vehicle Routing Problem

VRPTW Vehicle Routing Problem with Time Windows

GVNS General Variable Neighborhood Search

SGVNS Skewed General Variable Neighborhood Search

XML eXtensible Markup Language

DBMS database management system

POI Points of Interest

A* A-star

CG Column Generation

VRP Vehicle Routing Problem

UAV Unmanned Aerial Vehicles

PIH Pairwise Incentive Heuristic